

# Power Circuits, Exponential Algebra, and Time Complexity

Alexei G. Myasnikov, Alexander Ushakov, and Dong Wook Won

June 15, 2010

## Abstract

Motivated by algorithmic problems from combinatorial group theory we study computational properties of integers equipped with binary operations  $+$ ,  $-$ ,  $z = x2^y$ ,  $z = x2^{-y}$  (the former two are partial) and predicates  $<$  and  $=$ . Notice that in this case very large numbers, which are obtained as  $n$  towers of exponentiation in the base 2 can be realized as  $n$  applications of the operation  $x2^y$ , so working with such numbers given in the usual binary expansions requires super exponential space. We define a new compressed representation for integers by power circuits (a particular type of straight-line programs) which is unique and easily computable, and show that the operations above can be performed in polynomial time if the numbers are presented by power circuits. We mention several applications of this technique to algorithmic problems, in particular, we prove that the quantifier-free theories of various exponential algebras are decidable in polynomial time, as well as the word problems in some “hard to crack” one-relator groups.

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
1.1	Motivation . . . . .	2
1.2	Algorithmic problems for algebraic circuits . . . . .	4
1.3	Exponential algebras . . . . .	5
1.4	Our results . . . . .	6
1.5	Outline . . . . .	7
<b>2</b>	<b>Binary sums</b>	<b>9</b>
2.1	Elementary properties . . . . .	9
2.2	Shortest binary forms . . . . .	12
<b>3</b>	<b>Power circuits</b>	<b>14</b>
3.1	Power circuits and terms . . . . .	14
3.2	Term evaluation and constant circuits . . . . .	15

<b>4</b>	<b>Standard, reduced and normal power circuits</b>	<b>17</b>
4.1	Standard circuits . . . . .	17
4.2	Reduced power circuits . . . . .	20
4.3	Normal forms of constant power circuits . . . . .	21
<b>5</b>	<b>Reduction process</b>	<b>22</b>
5.1	Geometric order . . . . .	22
5.2	Equivalent vertices . . . . .	23
5.3	Reduction process . . . . .	29
<b>6</b>	<b>Computing normal forms</b>	<b>37</b>
<b>7</b>	<b>Elementary operations over power circuits</b>	<b>38</b>
7.1	Addition and subtraction . . . . .	38
7.2	Exponentiation . . . . .	39
7.3	Multiplication . . . . .	40
7.4	Multiplication and division by a power of two . . . . .	42
7.5	Ordering . . . . .	44
<b>8</b>	<b>Exponential algebra on power circuits</b>	<b>45</b>
8.1	Algebra of power circuits . . . . .	45
8.2	Power representation of integers . . . . .	47
8.3	Quantifier-free formulas in exponential algebra . . . . .	49
<b>9</b>	<b>Some inherent difficulties in computing with power circuits</b>	<b>50</b>
9.1	Division by 3 . . . . .	50
9.2	Power circuits and multiplication . . . . .	51
<b>10</b>	<b>Open Problems</b>	<b>51</b>

# 1 Introduction

In this paper we study power circuits (arithmetic circuits with exponentiation), and show that a number of algorithmic problems in algebra, involving exponentiation, is solvable in polynomial time.

## 1.1 Motivation

Massive numerical computations play a very important part in modern science. In one way or another they are usually reduced to computing with integers. This unifies various computational techniques over algebraic structures within the theory of constructive [19] or recursive models [23]. From a more practical view-point these reductions allow one to utilize the fundamental mathematical fact that the standard arithmetic manipulations with integers can be performed fairly quickly. In computations, integers are usually presented in the binary form, i.e., by words in the alphabet  $\{0,1\}$ . Given two integers  $a$

and  $b$  in the binary form one can perform the basic arithmetic operations in time  $O(N \log N \log \log N)$ , where  $N$  is the maximal binary length of  $a$  and  $b$  (see, for example, [4]). In the modern mathematical jargon one can say that the structure  $\mathbb{Z} = \langle \mathbb{Z}, +, -, \cdot, \leq \rangle$  (the standard arithmetic) is computable in at most quadratic time with respect to the binary representation of integers, or it is polynomial time computable (if we do not want to specify the degree of polynomials). This result holds for arbitrary  $n$ -ary representations of integers.

Notice, that the reductions mentioned above are not necessarily computable in polynomial time. In fact, there are recursive structures where polynomial time computations are impossible. Furthermore, there are many natural algebraic structures that admit efficient computations, though efficient algorithms are not easy to come by (we discuss some examples below). Usually, the core of the issue is to find a specific representation (data structure) of given algebraic objects which is suitable for fast computations.

For example, the standard representation of integer polynomials from  $\mathbb{Z}[X]$  as formal linear combinations of monomials in variables from  $X$ , may not be the most efficient way to compute with. Sometimes, it is more computationally advantageous to represent polynomials by *arithmetic circuits*. These circuits are finite directed labeled acyclic graphs  $C$  of a special type. Every node with non-zero in-degree in such a circuit  $C$  is labeled either by  $+$  (addition) or by  $-$  (subtraction), or by  $\cdot$  (multiplication); nodes of zero in-degree (source nodes) are labeled either by constants from  $\mathbb{Z}$  or by indeterminates  $x_i \in X$ . Going from the source nodes to a distinguished sink node (of zero out-degree) one can write down a polynomial  $p_C$  represented by the circuit  $C$ . Observe, that given two circuits  $C$  and  $D$  it is easy to construct a new circuit that represents the polynomial  $p_C + p_D$  (or  $p_C \cdot p_D$ ), so algebraic operations over circuits representations are almost trivial. In [30] Strassen used arithmetic circuits to design an efficient algorithm that performs matrix multiplication faster than  $O(n^3)$ . There are polynomial size circuits to compute determinants and permanents. We refer to a survey [31] and a book [7] for results on algebraic circuits and complexity.

The idea to use circuits (graphs) to represent terms of some fixed functional language, or the functions they represent, is rather general. For example, *boolean circuits* are used to deal with boolean formulas or functions. Here boolean formulas can be viewed as terms in the language  $\{\wedge, \vee, \neg, 0, 1\}$  of boolean algebras. Construction of boolean circuits is similar to the arithmetic ones, where the arithmetic operations are replaced by the boolean operations and integers are replaced by the constants 0, 1. Again, it is easy to define boolean operations over boolean circuits, but to check if two such circuits represent the same boolean function (or, equivalently, if a given circuit represents a satisfiable formula) is a much more difficult task (NP-hard). In 1949 Shannon suggested to use the size of a smallest circuit representing a given boolean function  $f$  as a measure of complexity of  $f$  [29]. Eventually, this idea developed into a major area of modern complexity theory, but this is not the main subject of our paper.

Another powerful application of the “circuit idea” is due to Plandowski, who introduced compression of words in a given finite alphabet  $X$  [21]. These

compressed words can be realized by circuits over the free monoid  $X^*$ , where the arithmetic operations are replaced by the monoid multiplication, so every such circuit  $C$  represents a word  $w_C \in X^*$ . The crucial point here is that the length of the word  $w_C$  can grow exponentially with the size of the circuit  $C$ , so the standard word algorithms become time-consuming. For instance, the direct algorithm to solve the comparison problem (if  $w_C = w_D$  for given circuits  $C, D$ ) requires exponential time, though there are smart polynomial time (in the size of the circuits) algorithms that can do that [21]. In [14] Lohrey proved similar results for reduced words in a given free group (in the group language). This brought a whole new host of efficient algorithms in group theory [28].

## 1.2 Algorithmic problems for algebraic circuits

In view of the examples above, we introduce here a general notion of an algebraic circuit and related algorithmic problems. Let  $\mathcal{L}$  be a finite set of symbols of operations (a functional language). An *algebraic circuit*  $C$  in  $\mathcal{L}$  (or an  $\mathcal{L}$ -circuit) is a finite directed graph whose nodes are either *input nodes* or *gates*. The input nodes have in-degree zero and are each labeled by either variables or constants from  $\mathcal{L}$ ; each gate is labeled by an operation from  $\mathcal{L}$  whose arity equals to the in-degree of the gate; vertices of out-degree zero are called *output nodes*. For a distinguished output vertex in  $C$  one can associate a term  $t_C$  as was described above. Observe, that this notion of an algebraic circuit is more general than the usual one (see, for example, [2, 31]), where  $\mathcal{L}$  is either the ring or field theory language. On the other hand, algebraic circuits can be viewed also as *straight-line programs* in  $\mathcal{L}$  (see [1] by Aho and Ullman). In our approach to algebraic circuits we follow [1], even though in this case the orientation of edges is reversed, but this should not confuse the reader.

There are several basic algorithmic problems associated with  $\mathcal{L}$ -circuits over a fixed algebraic structure  $A$  in  $\mathcal{L}$ . Denote by  $Const(A)$  the set of elements of  $A$  which are specified in  $\mathcal{L}$  as constants. The *value problem* (VP) is to find the value of the term  $t_C$  under an assignment of variables  $\eta : X \rightarrow Const(A)$  for a given  $\mathcal{L}$ -circuits  $C$ . The *value comparison problem* (VCP), mentioned above, is to decide if the terms  $t_C$  and  $t_D$  take the same value in  $A$  under the assignment  $\eta$  for given  $\mathcal{L}$ -circuits  $C$  and  $D$ . For a functional language  $\mathcal{L}$ , decidability of VCP in  $A$  implies decidability of the quantifier-free theory  $Th_{qf}(A)$  of  $A$ . More generally, one may allow any assignments  $\eta$  with values in a fixed subset  $S$  of  $A$ . In this case decidability of VCP in  $A$  relative to  $S$  is equivalent to decidability of the quantifier-free theory of  $A$  in a language  $\mathcal{L}_S$  (obtained from  $\mathcal{L}$  by adding constants from  $S$ ).

If the language  $\mathcal{L}$  contains predicates then decidability of  $Th_{qf}(A)$  depends completely on decidability of the set of atomic formulas in  $A$ . Recall, that atomic formulas in  $\mathcal{L}_S$  are of the form  $P(t_1^\eta, \dots, t_n^\eta)$ , where  $P$  is a predicate in  $\mathcal{L}$  (including equality) and  $t_i^\eta$  is evaluation of a term  $t_i$  under an assignment  $\eta$ . Notice, that if  $A$  is recursive then all the problems above are decidable in  $A$  in the language  $\mathcal{L}_A$ .

From now on we deal only with recursive structures, and our main concern

is the time complexity of the decision problems. This brings an important new twist to decision problems. It might happen that the direct evaluation of  $t_i$  under  $\eta$  is time consuming, so we prefer to keep  $t_i^\eta$  in the “compressed form”  $t_{C_i}$  for some  $\mathcal{L}$ -circuit  $C_i$  and proceed to checking whether or not the formula  $P(t_{C_1}, \dots, t_{C_n})$  holds in  $A$  without computing the values  $t_i^\eta$ . This is the essence of our approach to computational problems in this paper – we operate with terms  $t$  in their compressed form  $C_t$  to speed up computations. Such approach makes the following *term-realization problem* crucial: for a given term  $t(x_1, \dots, x_n)$  in  $\mathcal{L}$  construct in polynomial time an  $\mathcal{L}$ -circuit  $C$  such that  $t_C$  gives the function defined by  $t$  in  $A$ . A related *term-equivalence problem* asks for given  $\mathcal{L}$ -circuits  $C$  and  $D$  if the functions defined in  $A$  by  $t_C$  and  $t_D$  are equal or not. Observe, that  $t_C = t_D$  in  $A$  if and only if the identity  $\forall X(t_C(X) = t_D(X))$  holds in  $A$ . So decidability of the term-equivalence problem in  $A$  is equivalent to decidability of the equational theory of  $A$  (the set of all identities in  $\mathcal{L}_S$  which hold in  $A$ ).

### 1.3 Exponential algebras

In this paper we introduce and study algebraic circuits in exponential algebras. Typically, every such algebra has a unary exponential function  $y = E(x)$  as an operation, besides the standard ring operations of addition and multiplication. Some variations are possible here, so the language may contain additional operations (subtraction, division, multiplication by a power of 2, etc.) or predicates (ordering, divisibility, divisibility by a power of 2, etc.). We refer to such language, in all its incarnations, as to *exponential algebra* language and denote it by  $\mathcal{L}_{exp}$ . Exponential algebra is a very active part of modern algebra and model theory, it stems from two Tarski’s problems. The first one, The High School Algebra Problem, is about axioms of the equational theory of the high school arithmetic, i.e., the structure  $N_{HS} = \langle \mathbb{N}_{>0}; +, \cdot, x^y, 1 \rangle$ , where  $\mathbb{N}_{>0}$  is the set of positive integers. Namely, it asks if every identity that holds on  $N_{HS}$  logically follows from the classical “high school axioms” (introduced by Dedekind in [8]). This problem was settled in the negative by Wilkie in [34], where he gave an explicit counterexample. Moreover, it was shown that the equational theory of  $N_{HS}$  is not finitely axiomatizable, though decidable (Gurevich [10] and Macintyre [15]). The time complexity of the problem is unknown. Effective manipulations with terms over  $N_{HS}$  are important in numerous applications, it suffices to mention such programs as Mathematica, Maple, etc.

The second Tarski’s problem asks whether or not the elementary theory of the field of reals  $\mathbb{R}$  with the exponential function  $y = e^x$  in the language is decidable. In the paper [18] Macintyre and Wilkie proved that the elementary theory of  $(\mathbb{R}, e^x)$  is decidable provided the Schanuel’s Conjecture holds. The time complexity of the quantifier-free theory of  $(\mathbb{R}, e^x)$  or the term-equivalence problem for algebraic circuits over  $(\mathbb{R}, e^x)$  is unknown (see [24, 25, 26] for related problems).

## 1.4 Our results

Our main results here concern with the time complexity of the quantifier-free theory of the typical exponential algebras over natural numbers. We show that the quantifier-free theory is decidable in polynomial time in a structure  $\tilde{N} = \langle \mathbb{N}_{>0}; +, x \cdot 2^y, \leq, 1 \rangle$ , a slight modification of the high-school arithmetic  $N_{HS}$ , where, exponentiation and multiplication are replaced by  $x \cdot 2^y$  and the ordering predicate  $\leq$  is included. Of course, substituting 1 for  $x$  one gets the exponential function  $2^y$ . We show that the term-realization problem in  $\tilde{N}$  is decidable in polynomial time, as well as the quantifier-free theory  $Th_{qf}(\tilde{N})$ . This is precisely the case when the direct evaluation of a term for a particular assignment of variables might result in a superexponentially long number, so we avoid any direct evaluations of terms and work instead with algebraic circuits. The result holds if the partial function  $x \cdot 2^{-y}$  is added to the language. In this event for every quantifier-free sentence one can decide in polynomial time whether or not it holds in  $A$ , or is undefined. Strangely, the methods we exploit fail for the term-realization problem in the classical high-school arithmetic  $N_{HS}$ , the size of the resulting circuit may grow exponentially.

The Tarski's problem on decidability of  $(\mathbb{R}, e^x)$  generated very interesting research on exponential rings and fields (see, for example, [32, 17, 16, 33, 18, 35]). In [32, 16] a free commutative ring with exponentiation  $\mathbb{Z}[X]^E$  (with basis  $X$ ) was constructed – a free object in the variety of commutative unitary rings with an extra unitary operation for exponentiation  $y = E(x)$ . To perform various manipulations with exponential polynomials (elements of  $\mathbb{Z}[X]^E$ ) it is convenient to use power circuits, i.e., algebraic circuits over an algebraic structure  $\tilde{Z} = \langle \mathbb{Z}; +, -, x \cdot 2^y, \leq, 1 \rangle$ . The results described above for  $\tilde{N}$  hold also in  $\tilde{Z}$ , so the term-realization problem and the quantifier-free theory of  $\tilde{Z}$  are decidable in polynomial time. Whether these results hold with the multiplication in the language is an open problem.

In fact, our technique gives decidability in polynomial time of the term-realization problem and the quantifier-free theory of the classical exponential structures  $N_{HS}$  and  $Z_{exp} = \langle \mathbb{Z}; +, -, x \cdot y, 2^y, \leq, 1 \rangle$  even with the multiplication in the language if one considers only terms in the standard form, i.e., if they are given as exponential polynomials (see [32, 16]).

All the results mentioned above also hold if the exponentiation in the base 2 is replaced by an exponentiation in an arbitrary base  $n \in \mathbb{N}, n \geq 2$ . The argument for base 2 goes through in the general case as well.

Another application of power circuits comes from the theory of automatic structures, that was introduced by Hodgson [11], and Khoussainov and Nerode [13] (we refer to a recent survey [27] for details). Automatic structures form a nice subclass of recursive structures with decidable elementary theories. Arithmetic with weak division  $N_{weak} = \langle \mathbb{N}; S, +, \leq, |_2 \rangle$ , where  $x|_2 y$  if and only if  $x$  is a power of 2 and  $y$  is a multiple of  $x$  (“weak division”), is an important example of an automatic structure. It has the following universal property (see Blumensath and Gradel [5]): an arbitrary structure  $A$  has an automatic presentation if and only if it is interpretable (in model theory sense) in  $N_{weak}$ . This implies

that first-order questions about automatic structures can be reformulated as first-order questions on  $N_{weak}$ . It is known that the first order theory of  $N_{weak}$  is decidable, but its time complexity is non-elementary [5]. In view of the above, the complexity of the existential theory of  $N_{weak}$  is an open problem of prime interest. Notice, that complexity of the problem depends on the representation of the inputs. It follows from our results on power circuits that the quantifier-free theory of  $N_{weak}$  is decidable in polynomial time even when the numbers are presented in the compressed form by power circuits. We mention in passing that it would be interesting to see if the structure  $\tilde{N} = \langle \mathbb{N}_{>0}; +, x \cdot 2^y, \leq, 1 \rangle$  is automatic or not.

We would like to mention one more application of power circuits, which triggered this research in the first place. In the subsequent paper we use power circuits to solve a well-known open problem in geometric group theory. In 1969 Baumslag introduced ([3]) a one relator group

$$G = \langle a, b ; (b^{-1}ab)^{-1}a(b^{-1}ab) = a^2 \rangle,$$

which later became one of the most interesting examples in geometric group theory. It has been noticed by Gersten that the Dehn function of  $G$  cannot be bounded by any finite tower of exponents [9] (see complete proofs and upper bounds in the paper by Platonov [22]). The Word Problem in  $G$  is considered to be the hardest among all known one-relator groups. Recently, Kapovich and Schupp showed in [12] that the Word Problem in  $G$  is decidable in exponential time. Using power circuits we prove in [20] that the Word Problem in  $G$  is polynomial time decidable.

All the results above are based on a new representation of integers, which is much more “compressed” than the standard binary representation. This “power representation” is interesting in its own sake. We represent integers by constant power circuits in the normal form. Such representation is unique and easily computable: a number  $n \in \mathbb{N}$  can be presented by a normal power circuit  $\mathcal{P}_n$  of size at most  $\log_2 n + 2$ , and it takes time  $O(\log_2 n \log_2 \log_2 n)$  to find  $\mathcal{P}_n$ . Furthermore, we develop algorithms that allow one to perform the standard algebraic manipulations (in the structure  $\tilde{N}$ ) over integers given in power representation in polynomial time.

## 1.5 Outline

In Section 2 we introduce a new way to represent integers as binary sums (forms) by allowing coefficients  $-1$  in binary representations. In Section 2.1 we describe some elementary properties of these forms and design an algorithm that compares numbers given in such binary forms in linear time (in the size of the forms). In Section 2.2 we introduce “compact” binary sums which give shortest possible representations of numbers, and show that these forms are unique. It takes linear time (in the size of the standard binary representation) to compute the shortest binary form for a given integer  $n$ .

In Section 3 we give a definition of a general algebraic circuit in the language  $\mathcal{L} = \{+, -, \cdot, x \cdot 2^y\}$  and define a special type of circuits, called *power circuits*.



Power circuits are main technical objects of the paper. We show in due course that every algebraic circuit in  $\mathcal{L}$  is equivalent in the structure  $\tilde{Z} = \langle \mathbb{Z}, +, -, \cdot, x \cdot 2^y \rangle$  to a power circuit, but power circuits are much easier to work with. Besides, power circuits give a very compact presentation of natural numbers, designed specifically for efficient computations with exponential polynomials.

In Section 4 we define several important types of circuits: standard, reduced and normal. The standard ones can be easily obtained from general power circuits through some obvious simplifications. The reduced power circuits output numbers only, they require much stronger rigidity conditions (no redundant or superfluous pairs of edges, distinct vertices output distinct numbers), which are much harder to achieve. The normal power circuits are reduced and output numbers in the compact binary forms. They give a unique compact presentation of integers, which is much more compressed (in the worst case) than the canonical binary representations. This is the main construction of the paper, designed to speed up computations in exponential algebra. We hope that the construction is interesting in its own right.

In Section 5 we describe a reduction process which for a given constant power circuit  $\mathcal{P}$  constructs an equivalent reduced power circuit  $Reduce(\mathcal{P})$  in cubic time in the size of  $\mathcal{P}$ . This is the main technical result of the paper.

In Section 6 we show how to compute the normal power circuit representation of a given integer  $n$  (given in its binary representation) in time  $O(\log_2 n \log_2 \log_2 n)$ .

In Section 7 we describe how to perform the standard arithmetic operations and exponentiations (in the language  $\mathcal{L}$ ) over integers given in their power circuit representations. It turns out that the size of the resulting power circuits grows linearly, except for the ones produced by the multiplication (this is the main difficulty when dealing with power circuits). Finally, we show how to compare (in cubic time) the values of given constant power circuits without producing the binary representations of the actual numbers; and how to find the normal form of a given constant power circuit (in cubic time).

In Section 8 we solve some problems mentioned earlier in the introduction. Fix a language  $\mathcal{L} = \{+, -, *, x \cdot 2^y, x \cdot 2^{-y}, \leq, 0, 1\}$ , its sublanguage  $\mathcal{L}_0$ , which is obtained from  $\mathcal{L}$  by removing the multiplication  $*$ ; and structures  $\mathbb{Z}_{\mathcal{L}} = \langle \mathbb{Z}; +, -, *, x \cdot 2^y, x \cdot 2^{-y}, \leq, 1 \rangle$  and  $\tilde{Z} = \langle \mathbb{Z}; +, -, x \cdot 2^y, x \cdot 2^{-y}, \leq, 1 \rangle$ . We show that there exists an algorithm that for every algebraic  $L$ -circuit  $C$  finds an equivalent standard power circuit  $\mathcal{P}$ , or equivalently, there exists an algorithm which for every term  $t$  in the language  $\mathcal{L}$  finds a power circuit  $C_t$  which represents a term equivalent to the term  $t$  in  $\mathbb{Z}_{\mathcal{L}}$ . Moreover, if the term  $t$  is in the language  $\mathcal{L}_0$  then the algorithm computes the circuit  $C_t$  in linear time in the size of  $t$ . For integers and closed terms in  $\mathcal{L}_0$  one can get much stronger results. Let  $\mathcal{C}_{norm}$  be the set of all constant normal power circuits (up to isomorphism). We show that if  $t(X)$  is a term in  $\mathcal{L}_0$  and  $\eta : X \rightarrow \mathbb{Z}$  an assignment of variables, then there exists an algorithm which determines if  $t(\eta(X))$  is defined in  $\mathbb{Z}_{\mathcal{L}}$  (or  $\tilde{Z}$ ) or not; and if defined it then produces the normal circuit  $\mathcal{P}_t$  that presents the number  $t(\eta(X))$  in polynomial time. At the end of the section we prove that the quantifier-free theory of the structure  $\tilde{Z}$  with all the constants from  $\mathbb{Z}$  in the language is decidable in polynomial time.



In Section 9 we demonstrate some inherent difficulties when dealing with products of power circuits (the size of the resulting circuit grows exponentially).

Finally, in Section 10 we state some open problems on complexity of algorithms in the classical exponential algebras.

## 2 Binary sums

In this section we introduce a new way to represent integers as binary sums (forms) by allowing also coefficients  $-1$  in binary representations. In Section 2.1 we describe some elementary properties of these forms and design an algorithm that compares numbers given in such binary forms in linear time (in the size of the forms). In Section 2.2 we introduce “compact” binary sums which give shortest possible representations of numbers, and show that these forms are unique. It takes linear time (in the size of the standard binary representation) to compute the shortest binary form for a given integer  $n$ .

### 2.1 Elementary properties

A *binary term*  $P(\overline{x}, \overline{y})$  is a term in the language  $\{+, -, \cdot, 2^y\}$  (or  $\{+, -, x \cdot 2^y\}$ ) of the following type:

$$x_1 2^{y_1} + \dots + x_k 2^{y_k} \quad (\text{which we also denote by } \sum_{i=1}^k x_i 2^{y_i}). \quad (1)$$

Any assignment of variables  $x_i = \varepsilon_i, y_i = q_i$  with  $\varepsilon_i \in \{-1, 1\}$  and  $q_i \in \mathbb{N}$  ( $i = 1, \dots, k$ ) gives an algebraic expression, called a *binary sum* (or a *binary form*),

$$\varepsilon_1 2^{q_1} + \dots + \varepsilon_k 2^{q_k}, \quad (2)$$

which we also denote by  $\sum_{i=1}^k \varepsilon_i 2^{q_i}$  or  $P(\overline{\varepsilon}, \overline{q})$ , where  $\overline{\varepsilon} = (\varepsilon_1, \dots, \varepsilon_k)$   $\overline{q} = (q_1, \dots, q_k)$ . Let  $N(\overline{\varepsilon}, \overline{q})$  be the integer number resulting in performing all the operations in (2).

The standard binary representation of a natural number is a binary sum with  $\varepsilon_i \in \{0, 1\}$ . Every integer can be represented by infinitely many different binary sums. We say that two binary sums are *equivalent* if they represent the same number. Furthermore, a binary sum  $P(\overline{\varepsilon}, \overline{q})$  is *reduced* if the sequence  $\overline{q}$  is strictly decreasing. The following lemma is obvious.

**Lemma 2.1.** *The following hold:*

- 1) *For each binary sum  $P(\overline{\varepsilon}, \overline{q})$  there exists an equivalent reduced binary sum which can be computed in linear time  $O(|\overline{q}|)$ .*
- 2) *For any positive integer  $z$  there exists a unique reduced binary sum  $P(\overline{\varepsilon}, \overline{q})$  with  $\varepsilon_1 = \dots = \varepsilon_k = 1$  and  $q_k \leq \lfloor \log_2 z \rfloor$ , representing  $z$ . Furthermore, it can be found in  $O(\log_2 z)$  time.*

The unique binary sum representing a given natural number  $N$  with all coefficients  $\varepsilon_i = 1$  is called *positive normal form* of  $N$ .

**Remark 2.2.** Notice, that positive binary representations of numbers may not be the most efficient. For instance, the binary sum  $2^n - 2^0$  is equivalent to  $2^{n-1} + 2^{n-2} + \dots + 2^1 + 2^0$  but has much fewer terms.

**Lemma 2.3.** *Let  $P(\bar{\varepsilon}, \bar{q})$  be a reduced binary sum. Then:*

- 1)  $N(\bar{\varepsilon}, \bar{q}) = 0$  if and only if  $|\bar{q}| = 0$  (here  $|\bar{q}|$  is the length of the tuple  $\bar{q}$ ).
- 2)  $N(\bar{\varepsilon}, \bar{q}) > 0$  if and only if  $\varepsilon_1 = 1$ .
- 3)  $N(\bar{\varepsilon}, \bar{q}) < 0$  if and only if  $\varepsilon_1 = -1$ .
- 4)  $N(\bar{\varepsilon}, \bar{q})$  is divisible by  $2^n$  if and only if  $q_m \geq n$  (here  $m = |\bar{q}|$ , and  $n \in \mathbb{N}$ ).
- 5) In the notation above if  $N(\bar{\varepsilon}, \bar{q})$  is divisible by  $2^n$  and not divisible by  $2^{n+1}$  then  $q_m = n$ .

*Proof.* We prove 1), the rest is similar. If  $|\bar{q}| = 0$  then  $N(\bar{\varepsilon}, \bar{q}) = 0$ . Assume now that  $N(\bar{\varepsilon}, \bar{q}) = 0$  and  $\bar{q} = (q_1, \dots, q_k)$ , where  $k > 0$ . Let  $S = \{1 \leq i \leq k \mid \varepsilon_i > 0\}$ . Then

$$N(\bar{\varepsilon}, \bar{q}) = \left( \sum_{i \in S} 2^{q_i} \right) - \left( \sum_{j \in \{1, \dots, k\} \setminus S} 2^{q_j} \right).$$

The binary sums in the brackets have coefficients 1. Since  $P(\bar{\varepsilon}, \bar{q})$  is reduced these binary sums are different and by Lemma 2.1 define different numbers. This implies that  $N(\bar{\varepsilon}, \bar{q}) \neq 0$ , and 1) follows by contradiction.  $\square$

Let  $P(\bar{\varepsilon}, \bar{q})$  be a reduced binary sum. We say that a pair of powers  $(q_i, q_{i+1})$  in  $P(\bar{\varepsilon}, \bar{q})$  is *superfluous* if  $q_i = q_{i+1} + 1$  and  $\varepsilon_i = -\varepsilon_{i+1}$ . The next lemma shows that a binary sum with superfluous pairs can be simplified, by getting rid off such pairs in linear time.

**Lemma 2.4.** *Given a binary sum  $P(\bar{\varepsilon}, \bar{q})$  one can find an equivalent reduced binary sum without superfluous pairs in liner time  $O(|\bar{q}|)$ .*

*Proof.* Let  $(q_i, q_{i+1})$  be a superfluous pair in  $P(\bar{\varepsilon}, \bar{q})$ . Define

$$\bar{q}' = (q_1, \dots, q_{i-1}, q_{i+1}, \dots, q_n)$$

and

$$\bar{\varepsilon}' = (\varepsilon_1, \dots, \varepsilon_{i-1}, -\varepsilon_{i+1}, \dots, \varepsilon_n).$$

The equality  $\mp 2^{i+1} \pm 2^i = \mp 2^i$  implies  $N(\bar{\varepsilon}, \bar{q}) = N(\bar{\varepsilon}', \bar{q}')$ . Clearly, it requires linear number (in  $|\bar{q}|$ ) of steps like that to eliminate all superfluous pairs in  $P(\bar{\varepsilon}, \bar{q})$ .  $\square$

For a reduced binary sum  $\mathcal{P}(\overline{q}, \overline{\varepsilon})$  define

$$\varepsilon(\mathcal{P}, q) = \begin{cases} \varepsilon_j, & \text{if there exists (unique) } j \text{ such that } q_j = q; \\ 0, & \text{otherwise.} \end{cases}$$

The following technical lemma gives the main tool for efficiently comparing values of binary sums.

**Lemma 2.5.** *Let  $A = P(\overline{\varepsilon}, \overline{q})$  and  $B = P(\overline{\delta}, \overline{r})$  be reduced binary sums without superfluous pairs,  $k = |\overline{q}|$ , and  $m = |\overline{r}|$ . Put  $n = \max\{q_1, r_1\}$ ,  $\alpha_1 = \varepsilon(A, n)$ ,  $\alpha_2 = \varepsilon(A, n-1)$ ,  $\beta_1 = \varepsilon(B, n)$ , and  $\beta_2 = \varepsilon(B, n-1)$ . Then the following hold:*

- 1) *If  $\alpha_1 = 1$  and  $\beta_1 = -1$  then  $N(A) - N(B) \geq 2$ . Similarly, if  $\alpha_1 = -1$  and  $\beta_1 = 1$  then  $N(A) - N(B) \leq 2$ .*
- 2) *Assume  $\alpha_1 = 1$  and  $\beta_1 = 1$ , or  $\alpha_1 = -1$  and  $\beta_1 = -1$ . Define  $A' = P(\overline{\varepsilon}', \overline{q}')$  and  $B' = P(\overline{\delta}', \overline{r}')$ , where  $\overline{q}' = (q_2, \dots, q_k)$ ,  $\overline{\varepsilon}' = (\varepsilon_2, \dots, \varepsilon_k)$ ,  $\overline{r}' = (r_2, \dots, r_m)$ ,  $\overline{\delta}' = (\delta_2, \dots, \delta_m)$ . Then  $N(A) - N(B) = N(A') - N(B')$ .*
- 3) *Assume  $\alpha_1 = 1$  and  $\beta_1 = 0$ :*
  - a) *If  $\alpha_2 = 1$  then  $N(A) - N(B) \geq 2$ .*
  - b) *If  $\alpha_2 = 0$  and  $\beta_2 < 1$  then  $N(A) - N(B) \geq 2$ .*
  - c) *If  $\alpha_2 = 0$  and  $\beta_2 = 1$  define  $A' = P(\overline{\varepsilon}', \overline{q}')$  and  $B' = P(\overline{\delta}', \overline{r}')$ , where  $\overline{q}' = (n-1, q_2, \dots, q_k)$ ,  $\overline{\varepsilon}' = (1, \varepsilon_2, \dots, \varepsilon_k)$ ,  $\overline{r}' = (r_2, \dots, r_m)$ ,  $\overline{\delta}' = (\delta_2, \dots, \delta_m)$ . Then  $N(A) - N(B) = N(A') - N(B')$ .*

*Proof.* In the case 1) by Lemma 2.3  $N(A) \geq 1$  and  $N(B) \leq -1$ , so the statement holds. In the case 2)  $N(A') = N(A) - 2^n$  and  $N(B') = N(B) - 2^n$  and the statement holds. In the case 3.a)  $N(A) \geq 2^n + 2^{n-1} - 2^{n-2} + 1$  and  $N(B) \leq 2^n + 2^{n-1} - 1$  (since  $A$  and  $B$  have no superfluous pairs). In the case 3.b)  $N(A) \geq 2^{n-1} + 1$  and  $N(B) \leq 2^{n-1} - 1$ . In the case 3.c)  $N(A') = N(A) - 2^{n-1}$  and  $N(B') = N(B) - 2^{n-1}$ . These imply that 3) holds.  $\square$

**Proposition 2.6.** For given binary sums  $P(\overline{\varepsilon}, \overline{q})$  and  $P(\overline{\delta}, \overline{r})$  it takes linear time  $C(|\overline{q}| + |\overline{r}|)$  to compare the values  $N(\overline{\varepsilon}, \overline{q})$  and  $N(\overline{\delta}, \overline{r})$ .

*Proof.* By Lemmas 2.1 and 2.4 one can reduce and get rid off superfluous pairs in given binary sums in linear time. Now, let  $A = P(\overline{\varepsilon}, \overline{q})$  and  $B = P(\overline{\delta}, \overline{r})$  be reduced binary sums without superfluous pairs. In the notation of Lemma 2.5 one can describe the comparison algorithm as follows. Determine the values  $\alpha_1, \alpha_2$  and  $\beta_1, \beta_2$ . If they satisfy either of the case 1, 3.a, or 3.b then the answer follows immediately from the lemma. Otherwise, they satisfy either the case 2 or 3.c, and one can compute new binary sums  $A'$  and  $B'$  such that  $N(A') - N(B') = N(A) - N(B)$  and  $|A'| + |B'| < |A| + |B|$ , and compare their values. Notice that the binary sum  $A'$  in case 3.c) might contain a superfluous pair, which should be removed in the simplification process.

Now we describe the comparison algorithm formally.

**Algorithm 2.7.** (*To compare values of reduced binary sums with no superfluous pairs.*)

INPUT.  $P(\bar{\varepsilon}, \bar{q})$  and  $P(\bar{\delta}, \bar{r})$  two reduced binary sums with no superfluous pairs of powers.

OUTPUT.

$$\begin{cases} -2, & \text{if } N(\bar{\varepsilon}, \bar{q}) < N(\bar{\delta}, \bar{r}) - 1 \\ -1, & \text{if } N(\bar{\varepsilon}, \bar{q}) = N(\bar{\delta}, \bar{r}) - 1 \\ 0, & \text{if } N(\bar{\varepsilon}, \bar{q}) = N(\bar{\delta}, \bar{r}) \\ 1, & \text{if } N(\bar{\varepsilon}, \bar{q}) = N(\bar{\delta}, \bar{r}) + 1 \\ 2, & \text{if } N(\bar{\varepsilon}, \bar{q}) > N(\bar{\delta}, \bar{r}) + 1 \end{cases}$$

COMPUTATIONS.

- A) Remove all superfluous pairs from  $P(\bar{\varepsilon}, \bar{q})$  and  $P(\bar{\delta}, \bar{r})$ .
- B) Compute  $n = \max\{q_1, r_1\}$ .
- C) If  $n \leq 1$  then the current binary sums  $P(\bar{\varepsilon}, \bar{q})$  and  $P(\bar{\delta}, \bar{r})$  are at most one-bit numbers. Compute them, compare, and output the result.
- D) If  $n > 1$  then compute  $\alpha_1 = \varepsilon(P(\bar{\varepsilon}, \bar{q}), n)$ ,  $\alpha_2 = \varepsilon(P(\bar{\varepsilon}, \bar{q}), n - 1)$ ,  $\beta_1 = \varepsilon(P(\bar{\delta}, \bar{r}), n)$ , and  $\beta_2 = \varepsilon(P(\bar{\delta}, \bar{r}), n - 1)$ .
- E) Determine if  $(\alpha_1, \alpha_2)$  and  $(\beta_1, \beta_2)$  satisfy one of the cases 1, 3.a, or 3.b from Lemma 2.5. If so, return the result prescribed in Lemma.
- F) Determine if  $(\alpha_1, \alpha_2)$  and  $(\beta_1, \beta_2)$  satisfy one of the cases 2 or 3.c. If so, compute new binary sums  $A'$  and  $B'$  as prescribed in Lemma 2.5 put  $P(\bar{\varepsilon}, \bar{q}) = A'$  and  $P(\bar{\delta}, \bar{r}) = B'$  and goto A).

Notice, that each iteration of Algorithm 2.7 decreases the number  $|\bar{q}| + |\bar{r}|$  at least by 1, so the algorithm terminates in at most  $C(|\bar{q}| + |\bar{r}|)$  steps, as claimed.  $\square$

## 2.2 Shortest binary forms

Let  $P(\bar{\varepsilon}, \bar{q})$  be a reduced binary sum, where  $\bar{q} = (q_1, q_2, \dots, q_k)$ , and  $\bar{\varepsilon} = (\varepsilon_1, \dots, \varepsilon_k)$ . We say that  $P(\bar{\varepsilon}, \bar{q})$  is *compact* if  $q_{i+1} - q_i \geq 2$  for every  $i = 1, \dots, k - 1$ .

**Lemma 2.8.** *The following hold:*

- (1) *For any  $n \in \mathbb{N}$  there exists a unique compact binary sum  $P_n = \varepsilon_1 2^{q_1} + \dots + \varepsilon_k 2^{q_k}$  representing  $n$ . Furthermore,  $k, q_1, \dots, q_k \leq \log_2 n$  and  $P_n$  can be found in linear time  $O(\log_2 n)$ .*
- (2) *A compact binary sum representation of a given number involves the least possible number of terms.*
- (3) *Given a binary sum one can find an equivalent compact binary sum in linear time.*

*Proof.* By Lemma 2.1 for  $n \in \mathbb{N}$  we can find a reduced binary sum  $\mathcal{P}$  representing  $n$  in time  $O(\log_2 n)$ . Below we prove the existence and uniqueness of a compact binary sum equivalent to  $\mathcal{P}$ .

**Existence.** Consider any binary sum  $P(\bar{\varepsilon}, \bar{q})$ . Consider the following finite rewriting system  $\mathcal{C}$  on binary sums: a system of transformations of binary sums

$$\left\{ \begin{array}{l} 2^m + 2^m \rightarrow 2^{m+1} \\ 2^m - 2^m \rightarrow \varepsilon \\ 2^{m+1} + 2^m \rightarrow 2^{m+2} - 2^m \\ 2^{m+1} - 2^m \rightarrow 2^m \end{array} \right.$$

Obviously, each application of a rule from  $\mathcal{C}$  to a binary sum results in an equivalent binary sum, which is either shorter or has the same length as the initial sum. It is easy to see that the system  $\mathcal{C}$  is terminating, i.e., starting on a given binary sum  $P(\bar{\varepsilon}, \bar{q})$  after finitely many steps of rewriting one arrives to a sum that no rule from  $\mathcal{C}$  can be applied to. Observe, that the number of steps required here is at most linear in the length of  $P(\bar{\varepsilon}, \bar{q})$ . Furthermore, the system  $\mathcal{C}$  is locally confluent, hence confluent (see [6] for definitions). This implies that the rewriting of a given binary sum always results in a compact form and such a form does not depend on the rewriting process. In particular, applying the rewriting process to the standard binary representation of a given natural number  $n$  one can find the shortest binary form of  $n$  (and of  $-n$ ) in linear time.

**Uniqueness.** Consider two compact binary sums

$$P(\bar{\varepsilon}, \bar{q}) = \sum_{i=1}^k \varepsilon_i 2^{q_i}, \quad P(\bar{\delta}, \bar{p}) = \sum_{i=1}^s \delta_i 2^{p_i}.$$

Observe that

- If  $\varepsilon_k \neq \delta_s$  then  $N(\bar{\varepsilon}, \bar{q})$  and  $N(\bar{\delta}, \bar{p})$  have opposite signs, in particular  $N(\bar{\varepsilon}, \bar{q}) \neq N(\bar{\delta}, \bar{p})$ .

- If  $\varepsilon_k = \delta_s = 1$  and  $q_k > p_s$  then

$$N(\bar{\varepsilon}, \bar{q}) - N(\bar{\delta}, \bar{p}) \geq (2^{q_k} - 2^{q_k-2} - 2^{q_k-4} - \dots) - (2^{q_k-1} + 2^{q_k-3} + 2^{q_k-5} + \dots) \geq 1.$$

In particular  $N(\bar{\varepsilon}, \bar{q}) \neq N(\bar{\delta}, \bar{p})$ .

- Similarly,  $N(\bar{\varepsilon}, \bar{q}) \neq N(\bar{\delta}, \bar{p})$  whenever  $\varepsilon_k = \delta_s = -1$  and/or  $q_k < p_s$ .

Therefore, equality  $N(\bar{\varepsilon}, \bar{q}) = N(\bar{\delta}, \bar{p})$  implies that  $\varepsilon_k = \delta_s$  and  $q_k = p_s$ . Using this it is easy to prove that two compact binary sums representing the same number are equal.

**Minimality.** Any non-compact binary sum can be rewritten into an equivalent compact binary sum by the length non-increasing system  $\mathcal{C}$ . Therefore, the compact binary sums involve the least possible number of terms.  $\square$

**Lemma 2.9.** *Suppose  $P(\bar{\varepsilon}, \bar{q})$  is reduced and  $P(\bar{\delta}, \bar{p})$  is the equivalent compact binary sum. Then for every  $d \in \bar{p}$  either  $d \in \bar{q}$  or  $d - 1 \in \bar{q}$ . Furthermore, if  $N(\bar{\varepsilon}, \bar{q}) \neq 0$  then the compact binary sum representing the number  $N(\bar{\varepsilon}, \bar{q}) + 1$  satisfies the same condition.*

*Proof.* We may assume that  $P(\bar{\varepsilon}, \bar{q})$  does not contain superfluous pairs because removing superfluous pairs from  $P(\bar{\varepsilon}, \bar{q})$  results in a new binary sum  $P(\bar{\varepsilon}', \bar{q}')$  where  $\bar{q}' \subseteq \bar{q}$ . Therefore there exist sequences of positive integers  $\{a_i\}$ ,  $\{b_i\}$  and a sequence  $\{\varepsilon_i\}$  such that

$$P(\bar{\varepsilon}, \bar{q}) = (\varepsilon_1 2^{a_1} + \dots + \varepsilon_1 2^{a_1+b_1}) + \dots + (\varepsilon_k 2^{a_k} + \dots + \varepsilon_k 2^{a_k+b_k})$$

where  $a_i + b_i < a_{i+1}$  and  $\varepsilon_i = \pm 1$ . Making the sum in the first brackets compact we get

$$P(\bar{\varepsilon}, \bar{q}) = (-\varepsilon_1 2^{a_1} + \varepsilon_1 2^{a_1+b_1+1}) + \dots + (\varepsilon_k 2^{a_k} + \dots + \varepsilon_k 2^{a_k+b_k}).$$

If  $a_1 + b_1 + 1 \leq a_2 - 2$  then we can think that  $(-\varepsilon_1 2^{a_1} + \varepsilon_1 2^{a_1+b_1+1})$  is already compact and consider the next sum. The induction finishes the proof in this case.

Assume that  $a_1 + b_1 + 1 = a_2 - 1$ . If  $\varepsilon_1 = -\varepsilon_2$  then  $\varepsilon_1 2^{a_1+b_1+1} + \varepsilon_2 2^{a_2}$  is a superfluous pair. Removing it we obtain

$$P(\bar{\varepsilon}, \bar{q}) = (-\varepsilon_1 2^{a_1} - \varepsilon_1 2^{a_1+b_1+1}) + (\varepsilon_2 2^{a_2+1} + \dots + \varepsilon_2 2^{a_2+b_2}) + \dots$$

and as above the sum  $(-\varepsilon_1 2^{a_1} - \varepsilon_1 2^{a_1+b_1+1})$  is compact and we can consider the next sum. If  $\varepsilon_1 = \varepsilon_2$  then the power  $\varepsilon_1 2^{a_1+b_1+1}$  is being added to the second sum. Induction finishes the proof in this case.

Observe that in each case either we do not introduce a new power of 2 or we stop at  $2^{a_1+b_1+1}$ . Therefore, for every  $d \in \bar{p}$  either  $d \in \bar{q}$  or  $d - 1 \in \bar{q}$ . In a similar way we can prove the last statement of the lemma.  $\square$

### 3 Power circuits

We gave a definition of general algebraic circuits in the language  $\mathcal{L} = \{+, -, \cdot, x \cdot 2^y\}$  in the introduction. In this section we define a special type of circuits, called *power circuits*. Power circuits are main technical objects of the paper. They can be viewed as versions of the algebraic circuits of a special kind. We show in due course that every algebraic circuit in  $\mathcal{L}$  is equivalent in the structure  $\tilde{Z} = \langle \mathbb{Z}, +, -, \cdot, x \cdot 2^y \rangle$  to a power circuit, but power circuits are much easier to work with. Besides, power circuits give a very compact presentation of natural numbers, designed specifically for efficient computations with exponential polynomials.

#### 3.1 Power circuits and terms

Let  $\mathcal{P} = (V(\mathcal{P}), E(\mathcal{P}))$  be a directed graph. For an edge  $e = v_1 \rightarrow v_2 \in E(\mathcal{P})$  we denote by  $\alpha(e)$  its *origin*  $v_1$  and by  $\beta(e)$  its *terminus*  $v_2$ . We say that  $\mathcal{P}$

contains *multiple edges* if there are two distinct edges  $e_1$  and  $e_2$  in  $\mathcal{P}$  such that  $\alpha(e_1) = \alpha(e_2)$  and  $\beta(e_1) = \beta(e_2)$ . For a vertex  $v$  in  $\mathcal{P}$  denote by  $Out_v$  the set of all edges with the origin  $v$  and by  $In_v$  the set of all edges with the terminus  $v$ . A vertex  $v$  with  $Out_v = \emptyset$  is called a *leaf* or a *gate*;  $Leaf(\mathcal{P})$  is the set of leaves in  $\mathcal{P}$ .

A *power circuit* is a tuple  $(\mathcal{P}, \mu, M, \nu, \gamma)$  where:

- $\mathcal{P} = (V(\mathcal{P}), E(\mathcal{P}))$  is a non-empty directed acyclic graph with no multiple edges;
- $\mu : E(\mathcal{P}) \rightarrow \{1, -1\}$  is called the *edge labeling function*;
- $M \subseteq V(\mathcal{P})$  is a non-empty subset of vertices called *the marked vertices*;
- $\nu : M \rightarrow \{-1, 1\}$  is called a *sign function*.
- $\gamma : Leaf(\mathcal{P}) \rightarrow X \cup \{0\}$  is a function which assigns to each leaf in  $\mathcal{P}$  either a variable from a set of variables  $X$  or the constant 0.

For simplicity we often omit  $\mu, M, \nu, \gamma$  from notation and refer to the power circuit above as  $\mathcal{P}$ .

For a power circuit  $\mathcal{P}$  we define a term  $t_v$  in the language  $\mathcal{L}$  for each vertex  $v \in V(\mathcal{P})$ , by induction starting at leaves (which exists since  $\mathcal{P}$  is acyclic):

$$t_v = \begin{cases} \gamma(v) & \text{if } v \in Leaf(\mathcal{P}); \\ 2^{\sum_{e \in Out_v} \mu(e)t_v(\beta(e))} & \text{otherwise.} \end{cases}$$

where the sum  $\sum_{e \in Out_v}$  denotes composition of additions in some fixed order on terms in  $\mathcal{L}$ . Finally, define a term

$$\mathcal{T}_{\mathcal{P}} = \sum_{v \in M} \nu(v)t_v.$$

The number  $|\mathcal{P}| = |V(\mathcal{P})| + |E(\mathcal{P})|$  is called the *size* of a power circuit  $\mathcal{P}$ . Two circuits  $\mathcal{P}_1$  and  $\mathcal{P}_2$  are *equivalent* (symbolically  $\mathcal{P}_1 \sim \mathcal{P}_2$ ) if the terms  $\mathcal{T}_{\mathcal{P}_1}$  and  $\mathcal{T}_{\mathcal{P}_2}$  induce the same function in  $\tilde{Z}$ . Notice, that these functions could be partial (not everywhere defined) on  $\tilde{Z}$ .

### 3.2 Term evaluation and constant circuits

A power circuit  $\mathcal{P} = (\mathcal{P}, \mu, M, \nu, \gamma)$  is called *constant* if the function  $\gamma$  assigns no variables (i.e.,  $\gamma \equiv 0$ ). In this case every term  $t_v$  ( $v \in V(\mathcal{P})$ ), represents a real number which we denote by  $\mathcal{E}(v)$ . The real represented by  $\mathcal{T}_{\mathcal{P}}$  is denoted by  $\mathcal{E}(\mathcal{P})$ . We say that  $\mathcal{P}$  *properly represents* an integer number  $N$  if  $N = \mathcal{E}(\mathcal{P})$  and  $\mathcal{E}(v) \in \mathbb{N}$  for every  $v \in V(\mathcal{P})$ . In this case we write  $N = \mathcal{E}(\mathcal{P})$ . Notice that the term  $\mathcal{T}_{\mathcal{P}}(X)$  is defined in  $\tilde{Z}$  for an assignment of variables  $\eta : X \rightarrow \mathbb{Z}$  if and only  $\mathcal{P}$  properly represent an integer  $N(\mathcal{P})$ . Similarly, we say that  $\mathcal{P}$  properly represents a natural number if  $\mathcal{E}(\mathcal{P}) \in \mathbb{N}$  and  $\mathcal{E}(v) \in \mathbb{N}$  for every  $v \in V(\mathcal{P})$ . Observe, that two constant circuits  $\mathcal{P}_1$  and  $\mathcal{P}_2$  are *equivalent* if  $\mathcal{E}(\mathcal{P}_1) = \mathcal{E}(\mathcal{P}_2)$ .



For constant circuits we omit the function  $\gamma$  from notation. Equivalent power circuits  $\mathcal{P}_1$  and  $\mathcal{P}_2$  are *strongly equivalent* if  $\mathcal{P}_1$  is proper if and only if  $\mathcal{P}_2$  is proper.

**Lemma 3.1.** *For  $n \in \mathbb{N}$  one constructs a power circuit  $\mathcal{P}$  properly representing  $n$  in time  $O(\log_2^2 n)$ .*

*Proof.* Induction on  $n$ . By Lemma 2.1 for a given number  $n$  one can find in time  $O(\log_2 n)$  the reduced binary sum  $\varepsilon_1 2^{q_1} + \dots + \varepsilon_k 2^{q_k}$  representing  $n$ , where  $k, q_1, \dots, q_k \leq \log_2 n$ . By induction, one can construct power circuits  $C_1, \dots, C_k$  representing the numbers  $q_1, \dots, q_k$ . It takes time  $O(\log_2^2(\log_2 n))$  to construct each circuit  $C_i$ . So altogether it takes at most  $O(\log_2 n \log_2^2(\log_2 n))$  time. Given power circuits  $C_1, \dots, C_k$  it requires additional time  $O(\log_2 n)$  to construct a power circuit representing  $n$ . The time estimate follows from the obvious observation  $\log_2 n \log_2^2(\log_2 n) = O(\log_2^2 n)$ .  $\square$

See Figure 1 for examples of (constant) power circuits. In figures we denote unmarked vertices by white circles and marked vertices by black circles. Each edge and marked vertex is labelled with the plus or minus sign denoting 1 or  $-1$  respectively.

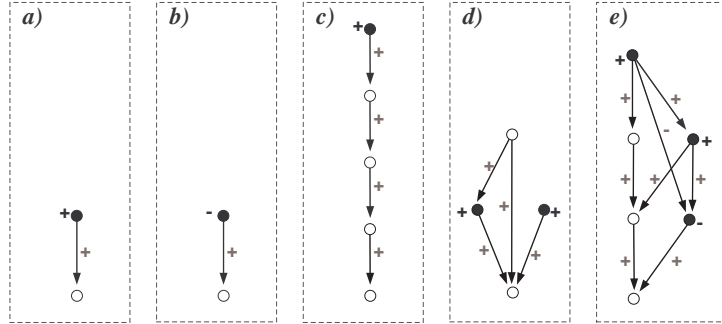


Figure 1: Examples of power circuits representing integers 1,  $-1$ , 16, 2, and 35.

Let  $\mathcal{P}$  be a power circuit,  $\mathcal{T}_{\mathcal{P}} = \mathcal{T}_{\mathcal{P}}(X)$ , and  $\eta : X \rightarrow \mathbb{Z}$  an assignment of variables in  $X$ . The following lemma allows one to operate with the value  $\mathcal{T}_{\mathcal{P}}(\eta(X))$  by means of constant power circuits.

**Lemma 3.2.** *Let  $\mathcal{P}$  be a power circuit,  $\mathcal{T}_{\mathcal{P}} = \mathcal{T}_{\mathcal{P}}(X)$ , and  $\eta : X \rightarrow \mathbb{Z}$  an assignment of variables. Then one can construct a constant power circuit  $\mathcal{P}'$  representing the number  $\mathcal{T}_{\mathcal{P}}(\eta(X))$  in time  $O(|\mathcal{P}| + \log_2^2(\text{size}(\eta)))$ , where  $\text{size}(\eta) = \sum_{x \in X} \log_2(\eta(x))$ . Moreover, the value  $\mathcal{T}_{\mathcal{P}}(\eta(X))$  is defined in  $\tilde{Z}$  if and only if the circuit  $\mathcal{P}'$  properly represents  $\mathcal{T}_{\mathcal{P}}(\eta(X))$ .*

## 4 Standard, reduced and normal power circuits

In this section we define several important types of circuits: standard, reduced and normal. The standard ones can be easily obtained from general power circuits through some obvious simplifications. The reduced power circuits output numbers only, they require much stronger rigidity conditions (no redundant or superfluous pairs of edges, distinct vertices output distinct numbers), which are much harder to achieve. The normal power circuits are reduced and output numbers in the compact binary forms. They give a unique compact presentation of integers, which is much more compressed (in the worst case) than the canonical binary representations. This is the main construction of the paper, it is interesting in its own right.

### 4.1 Standard circuits

We say that a vertex  $v$  is a *zero vertex* in a circuit  $\mathcal{P}$  if  $v \in \text{Leaf}(\mathcal{P})$  and  $\gamma(v) = 0$ . If  $\mathcal{P}$  is a constant circuit then  $v$  is a zero vertex if and only if  $\mathcal{E}(v) = 0$ . The following lemma is obvious.

**Lemma 4.1.** *A constant power circuit contains at least one zero vertex.*

Let  $\mathcal{P}$  be a constant power circuit. Below we describe some obvious *rewriting rules* that allow one to simplify  $\mathcal{P}$  (if applicable) keeping the strong equivalence.

**Trivializing:** Notice that if every marked vertex in  $\mathcal{P}$  is a zero vertex then  $\mathcal{E}(\mathcal{P}) = 0$ . In this event we replace  $\mathcal{P}$  by a strongly equivalent circuit  $\mathcal{P}'$  consisting of a single marked vertex  $v$ .

From now on we assume that  $\mathcal{P}$  has a non-zero marked vertex.

**Unmark a zero:** Let  $v$  be a marked zero vertex in  $\mathcal{P}$ . If  $\mathcal{P}'$  is obtained from  $\mathcal{P}$  by making  $v$  unmarked then  $\mathcal{P}$  and  $\mathcal{P}'$  are strongly equivalent.

**Fold two zeros:** Let  $v_1, v_2 \in V(\mathcal{P})$  be two zeros in  $\mathcal{P}$ . If  $\mathcal{P}'$  is obtained from  $\mathcal{P}$  by folding  $v_1$  and  $v_2$  then  $\mathcal{P}$  and  $\mathcal{P}'$  are strongly equivalent.

**Remove redundant “zero edges”:** Let  $z$  be a zero vertex in  $\mathcal{P}$ ,  $e = v \rightarrow z \in E(\mathcal{P})$ , and  $|\text{Out}_v| > 1$ . If  $\mathcal{P}'$  is obtained from  $\mathcal{P}$  by removing the edge  $e$  then  $\mathcal{P}$  and  $\mathcal{P}'$  are strongly equivalent.

A power circuit  $\mathcal{P}$  is *trimmed* if for each vertex  $v \in \mathcal{P}$  there is a directed path from a marked vertex to  $v$ . The following rewriting rule allows one to trim circuits.

**Trimming:** Let  $v$  be an unmarked vertex  $v \in \mathcal{P}$  with  $\text{In}_v = \emptyset$ . If  $\mathcal{P}'$  is obtained from  $\mathcal{P}$  by removing the vertex  $v$  and all the adjacent edges then  $\mathcal{P}'$  is strongly equivalent to  $\mathcal{P}$ .

**Definition 4.2.** A trimmed power circuit  $\mathcal{P}$  is in the *standard* form if it contains a unique unmarked zero vertex and contains no redundant zero edges.

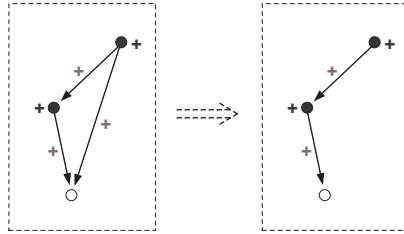


Figure 2: Removing redundant zero edges.

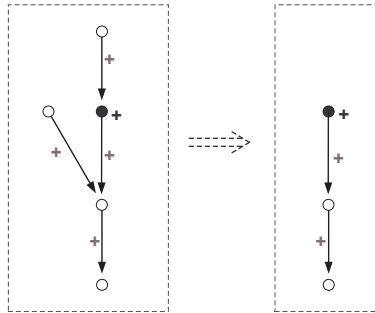


Figure 3: Example of trimming.

**Algorithm 4.3.** (*Standard power circuit*)

INPUT. A circuit  $\mathcal{P}$ .

OUTPUT. A strongly equivalent circuit  $\mathcal{P}'$  in a standard form.

COMPUTATIONS.

- (1) Compute the set  $Leaf(\mathcal{P})$ .
- (2) Fold all zero vertices in  $Leaf(\mathcal{P})$  into one vertex  $z$  and make it unmarked.
- (3) Erase all redundant edges incoming into  $z$ .
- (4) Trim the circuit.
- (5) Return the result  $\mathcal{P}'$ .

Summarizing the argument above one has the following result.

**Proposition 4.4.** Let  $\mathcal{P}'$  be produced from  $\mathcal{P}$  by Algorithm 4.3. Then

- $\mathcal{P}'$  is standard and is strongly equivalent to  $\mathcal{P}$ .
- $|V(\mathcal{P})| \leq |V(\mathcal{P}')|$  and  $|E(\mathcal{P})| \leq |E(\mathcal{P}')|$ .
- it takes time  $O(|\mathcal{P}|)$  to construct  $\mathcal{P}'$ .

There is one more procedure that is useful for operations over power circuits (see Sections 7.3 and 7.4). Recall that a vertex  $v$  in  $\mathcal{P}$  is a *source* if  $In_v = \emptyset$ . The following algorithm converts a circuit into an equivalent one where each marked vertex is a source.

**Algorithm 4.5.**

INPUT. A circuit  $\mathcal{P} = (\mathcal{P}, M, \mu, \nu)$ .

OUTPUT. An equivalent circuit  $\mathcal{P}'$  in which every marked vertex is a source.

COMPUTATIONS:

- A. For each vertex  $v \in M$  with  $In_v \neq \emptyset$  do:
  - (1) introduce a new vertex  $v'$ ;
  - (2) for each edge  $v \xrightarrow{\varepsilon} u$  introduce a new edge  $v' \xrightarrow{\varepsilon} u$ ;
  - (3) replace  $v$  with  $v'$  in  $M$  and put  $\nu(v') = \nu(v)$ .
- B) Output the obtained circuit.

**Lemma 4.6.** Let  $\mathcal{P}'$  be produced by Algorithm 4.5 from  $\mathcal{P}$ . Then:

- $\mathcal{P}$  is strongly equivalent to  $\mathcal{P}'$ ,
- $|V(\mathcal{P}')| \leq 2|V(\mathcal{P})|$ , and  $|E(\mathcal{P}')| \leq 2|E(\mathcal{P})|$ .
- Algorithm 4.5 has linear time complexity  $O(|\mathcal{P}|)$ .

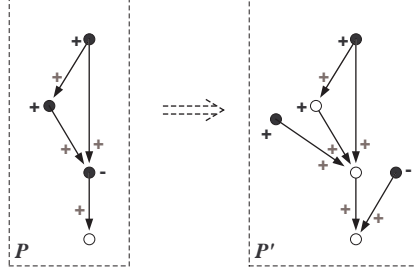


Figure 4: Processing of marked vertices that are not sources.

## 4.2 Reduced power circuits

Let  $\mathcal{P}$  be a constant power circuit in the standard form. A pair of edges  $e_1 = v \rightarrow v_1$  and  $e_2 = v \rightarrow v_2$  with the same origin  $v$  is called a *redundant* pair in  $\mathcal{P}$  if  $\mu(e_1) = -\mu(e_2)$  and  $\mathcal{E}(v_1) = \mathcal{E}(v_2)$ .

**Removing redundant edges:** Let  $e_1$  and  $e_2$  be a redundant pair of edges in  $\mathcal{P}$ . If  $\mathcal{P}'$  is obtained from  $\mathcal{P}$  by removing the pair  $e_1, e_2$  then  $\mathcal{P}'$  is equivalent to  $\mathcal{P}$ . Moreover, if  $\mathcal{P}$  properly represents an integer then  $\mathcal{P}'$  properly represents the same integer.

A pair of edges  $e_1 = v \rightarrow v_1$  and  $e_2 = v \rightarrow v_2$  as above is termed *superfluous* if  $\mu(e_1) = -\mu(e_2)$  and  $\mathcal{E}(v_1) = 2\mathcal{E}(v_2)$ .

**Removing superfluous edges:** Let  $(e_1, e_2)$  be a pair of superfluous edges in  $\mathcal{P}$ . If  $\mathcal{P}'$  is obtained from  $\mathcal{P}$  by removing the edge  $e_1$  from  $\mathcal{P}$  and changing  $\mu(e_2)$  to  $-\mu(e_2)$  then  $\mathcal{P}'$  is strongly equivalent to  $\mathcal{P}$ .

**Remark.** If one knows what pairs of edges are redundant or superfluous in  $\mathcal{P}$  then it takes time  $O(|\mathcal{P}|)$  to remove them (applying the rules above). However, it is not obvious how to check efficiently if  $\mathcal{E}(v_1) = \mathcal{E}(v_2)$  or  $\mathcal{E}(v_1) = 2\mathcal{E}(v_2)$ . We take care of this in due course.

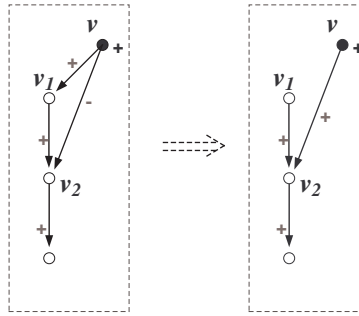


Figure 5: Removing superfluous edges. Here,  $\mathcal{E}(v_1) = 2\mathcal{E}(v_2) = 2$ .

**Definition 4.7.** A circuit  $\mathcal{P}$  is *reduced* if

- (R1)  $\mathcal{P}$  is in the standard form.
- (R2) For any  $v_1, v_2 \in V(\mathcal{P})$ ,  $\mathcal{E}(v_1) = \mathcal{E}(v_2)$  if and only if  $v_1 = v_2$ .
- (R3)  $\mathcal{P}$  contains no redundant or superfluous edges.

**Proposition 4.8.** Let  $\mathcal{P}$  be a reduced circuit. Then

- 1)  $\mathcal{E}(\mathcal{P}) = 0$  if and only if  $\mathcal{P}$  is trivial, i.e.,  $\mathcal{P}$  consists of a single marked vertex.
- 2) Let  $v \in M$  be such that for any  $v' \in M$   $\mathcal{E}(v) \geq \mathcal{E}(v')$  (the vertex with the maximal  $\mathcal{E}$ -value in  $M$ ). Then:
  - If  $\nu(v) = 1$  then  $\mathcal{E}(\mathcal{P}) > 0$ .
  - If  $\nu(v) = -1$  then  $\mathcal{E}(\mathcal{P}) < 0$ .

*Proof.* If  $\mathcal{P}$  is trivial then clearly  $\mathcal{E}(\mathcal{P}) = 0$ . Now, suppose that  $\mathcal{P}$  is not trivial. Then  $\mathcal{E}(\mathcal{P}) = \sum_{v \in M} \nu(v) \mathcal{E}(v)$ , where  $\mathcal{E}(v) = 2^{\sum_{e \in \text{Out}_v} \mu(e) \mathcal{E}(\beta(e))}$ . Hence  $\mathcal{E}(\mathcal{P})$  is a reduced binary sum, so by Lemma 2.3, it is not equal to 0. Which proves 1).

The second statement can be proved similarly using Lemma 2.3. □

### 4.3 Normal forms of constant power circuits

Let  $\mathcal{P}$  be a constant power circuit. We say that  $\mathcal{P}$  is in the *normal form* if

- (N1)  $\mathcal{P}$  is proper and reduced.
- (N2) For every vertex  $v \in V(\mathcal{P})$  the binary sum  $\sum_{e \in \text{Out}_v} \mu(e) \mathcal{E}(\beta(e))$  is compact (after proper enumeration of children of  $v$ ).
- (N3) The binary sum  $\mathcal{E}(\mathcal{P}) = \sum_{v \in M} \nu(v) \mathcal{E}(v)$  is in the compact form.

Power circuits  $\mathcal{P}_1$  and  $\mathcal{P}_2$  are *isomorphic* if there exists a graph isomorphism  $\varphi : \mathcal{P}_1 \rightarrow \mathcal{P}_2$  mapping  $M(\mathcal{P}_1)$  bijectively onto  $M(\mathcal{P}_2)$  and preserving the values of  $\mu$ ,  $\nu$ , and  $\gamma$ .

**Theorem 4.9.** *Two constant power circuits in the normal form are equivalent if and only if they are isomorphic.*

*Proof.* “ $\Leftarrow$ ” Obvious.

“ $\Rightarrow$ ” For  $v_1 \in V(\mathcal{P}_1)$  we define  $\varphi(v_1)$  to be the vertex  $v_2 \in V(\mathcal{P}_2)$  such that  $\mathcal{E}(v_1) = \mathcal{E}(v_2)$ . Below we prove that for every  $v_1$  there exists  $v_2$  with that property. Uniqueness of  $v_2$  follows from the fact that  $\mathcal{P}_2$  is reduced.

Since  $\mathcal{P}_1$  and  $\mathcal{P}_2$  are equivalent we have

$$\sum_{v \in M(\mathcal{P}_1)} \nu(v) \mathcal{E}(v) = \mathcal{E}(\mathcal{P}_1) = \mathcal{E}(\mathcal{P}_2) = \sum_{v \in M(\mathcal{P}_2)} \nu(v) \mathcal{E}(v),$$

where  $\mathcal{E}(v) = 2^{\sum_{e \in \text{Out}_v} \mu(e) \mathcal{E}(\beta(e))}$ . By (N3) the sums for  $\mathcal{E}(\mathcal{P}_1)$  and  $\mathcal{E}(\mathcal{P}_2)$  are compact and, hence, by Lemma 2.8 are essentially the same (up to a permutation of summands). Therefore,  $\varphi$  defined above gives a one to one correspondence between  $M(\mathcal{P}_1)$  and  $M(\mathcal{P}_2)$ .

Suppose that  $v_1 \in V(\mathcal{P}_1)$  and  $v_2 \in V(\mathcal{P}_2)$  satisfy  $\mathcal{E}(v_1) = \mathcal{E}(v_2)$ . Then

$$\sum_{e \in \text{Out}_{v_1}} \mu(e) \mathcal{E}(\beta(e)) = \sum_{e \in \text{Out}_{v_2}} \mu(e) \mathcal{E}(\beta(e))$$

and both sums are in compact form by (N2). By Lemma 2.8 these sums are essentially the same and there is one to one correspondence of the summands.

Finally, since  $\mathcal{P}_1$  and  $\mathcal{P}_2$  are trimmed, every vertex is a descendant of a marked vertex. Therefore, we can inductively extend the one to one correspondence  $\varphi$  from the marked vertices to all vertices of  $\mathcal{P}_1$ . It is easy to see that  $\varphi$  is a required graph isomorphism preserving values of  $\mu$ ,  $\nu$ , and  $\gamma$ .  $\square$

## 5 Reduction process

The main goal of this section is to prove the following theorem, which is the main technical result of the paper.

**Theorem 5.1.** *There is an algorithm that given a constant power circuit  $\mathcal{P}$  constructs an equivalent reduced power circuit  $\mathcal{P}'$  in time  $O(|V(\mathcal{P})|^3)$ . Moreover,  $|V(\mathcal{P}')| \leq |V(\mathcal{P})| + 1$ .*

We accomplish this in a series of lemmas and propositions. The algorithm itself is described as Algorithm 5.14 below.

### 5.1 Geometric order

In this section we present an algorithm which transforms a circuit  $\mathcal{P}$  into a reduced one. Property (R1) and (R3) can be easily achieved using Algorithm 4.3 which produces a trimmed strongly equivalent standard circuit of smaller size. Our main goal is to find an algorithm that produces equivalent circuit satisfying property (R2).

We say that a sequence  $\{v_1, \dots, v_n\}$  of vertices of  $\mathcal{P}$  is *geometrically ordered* if for each edge  $e = v_i \rightarrow v_j \in E(\mathcal{P})$  we have  $i > j$ .

**Lemma 5.2.** *For any circuit  $\mathcal{P}$  there exists a geometric order on  $V(\mathcal{P})$ .*

*Proof.* Induction on the number of vertices. Clearly a geometric ordering exists for  $\mathcal{P}$  with  $|V(\mathcal{P})| = 1$ . Assume it exists for any directed graph  $\mathcal{P}$  without loops such that  $|V(\mathcal{P})| < N$ . Let  $\mathcal{P}$  be a graph on  $N$  vertices. Then by Lemma 4.1 there is a zero vertex  $z$  in  $\mathcal{P}$ . Let  $\mathcal{P}'$  be obtained from  $\mathcal{P}$  by removing  $z$  and  $\{v_1, \dots, v_{N-1}\}$  be a geometric order of its vertices. Then clearly  $\{z, v_1, \dots, v_{N-1}\}$  is a geometric order on  $V(\mathcal{P})$ .  $\square$



**Lemma 5.3.** *Assume that  $\{v_1, \dots, v_n\}$  is a geometric order on  $V(\mathcal{P})$ . If  $|V(\mathcal{P})| \geq 1$  then  $v_1$  is a zero in  $\mathcal{P}$ . If  $|V(\mathcal{P})| \geq 2$  and  $\mathcal{P}$  has the unique zero then  $\mathcal{E}(v_2) = 1$ .*

*Proof.* Clearly  $Out_{v_1} = \emptyset$ . Otherwise, there is  $v \in Out_{v_1}$  and by definition of geometric order,  $v_1$  cannot precede  $v$  which gives a contradiction.

If  $\mathcal{P}$  has a unique zero and  $|V(\mathcal{P})| \geq 2$  then  $Out_{v_2} \neq \emptyset$ . The only edge in  $Out_{v_2}$  must be  $(v_2, v_1)$  (otherwise we get a contradiction with geometric order). Hence  $\mathcal{E}(v_2) = 1$ . □

## 5.2 Equivalent vertices

In this section we define an inductive step for reduction of power circuits. Let  $\mathcal{P}$  be a power circuit. We say that vertices  $v_i$  and  $v_j$  are called *equivalent* if  $\mathcal{E}(v_i) = \mathcal{E}(v_j)$ . Clearly,  $\mathcal{P}$  satisfies (R2) if and only if it does not contain equivalent vertices.

Assume that  $\mathcal{P}$  is a circuit satisfying (R1) and (R3) and  $v_i, v_j$  is the only pair of distinct equivalent vertices in  $\mathcal{P}$ . **All circuits in this section are of this type.** In this section we show how one can double the value of  $\mathcal{E}(v_j)$  in  $\mathcal{P}$  while keeping  $\mathcal{E}$ -values of all other vertices and the value  $\mathcal{E}(\mathcal{P})$  the same. Using that algorithm we show later how one can obtain a reduced circuit  $\mathcal{P}'$  equivalent to  $\mathcal{P}$ . The next algorithm transforms the given circuit  $\mathcal{P}$  so that the vertices  $v_i, v_j$  are not reachable from each other along directed paths.

### Algorithm 5.4.

INPUT. A circuit  $\mathcal{P}$  satisfying the properties of this section.

OUTPUT. An equivalent circuit  $\mathcal{P}'$  satisfying the properties of this section such that  $v_i$  and  $v_j$  are not reachable from each other.

COMPUTATIONS.

A) If  $v_i$  is reachable from  $v_j$  then:

- (1) Remove all edges leaving  $v_j$ .
- (2) For each edge  $v_i \xrightarrow{\mu} u$  add an edge  $v_j \xrightarrow{\mu} u$ .
- (3) Output the obtained circuit.

B) If  $v_j$  is reachable from  $v_i$  then perform steps as in the case A. for  $v_i$  and output the result.

C) If neither of  $v_i, v_j$  is reachable from the other then output  $\mathcal{P}$ .

Observe that Algorithm 5.4 does not change the vertex set of  $\mathcal{P}$ . In the next lemma we prove that the output of Algorithm 5.4 possesses all the claimed properties.

**Lemma 5.5.** *Let  $\mathcal{P}$  be a circuit on vertices  $\{v_1, \dots, v_n\}$ ,  $\mathcal{P}'$  the output of Algorithm 5.4, and  $V(\mathcal{P}') = \{v'_1, \dots, v'_n\}$  where  $v'_i \in V(\mathcal{P}')$  corresponds to  $v_i \in V(\mathcal{P})$ . Let  $v_i$  and  $v_j$  be two distinct vertices with  $\mathcal{E}(v_i) = \mathcal{E}(v_j)$ . Then*

- 1)  $\mathcal{E}(\mathcal{P}) = \mathcal{E}(\mathcal{P}')$ .
- 2)  $\mathcal{E}(v_i) = \mathcal{E}(v'_i)$  for every  $k = 1, \dots, n$ .
- 3) Neither  $v_i$  nor  $v_j$  is reachable from the other through directed edges in  $\mathcal{P}'$

Moreover, it takes linear time  $O(|\mathcal{P}|)$  to construct  $\mathcal{P}'$ .

*Proof.* Assume that  $v_i$  is reachable from  $v_j$  in  $\mathcal{P}$  along a directed path. By assumption of the lemma we have

$$\mathcal{E}(v_j) = 2^{\left(\sum_{e \in \text{Out}_{v_j}} \mu(e) \mathcal{E}(\beta(e))\right)} = 2^{\left(\sum_{e \in \text{Out}_{v_i}} \mu(e) \mathcal{E}(\beta(e))\right)} = \mathcal{E}(v_i)$$

and, therefore,  $\sum_{e \in \text{Out}_{v_i}} \mu(e) \mathcal{E}(\beta(e)) = \sum_{e \in \text{Out}_{v_j}} \mu(e) \mathcal{E}(\beta(e))$ . Thus, replacing edges leaving  $v_j$  with edges leaving  $v_i$  does not change  $\mathcal{E}(v_j)$ . Furthermore, it is easy to show that  $\mathcal{E}$ -values of all other vertices do not change. Finally, since the sign function does not change the obtained circuit is equivalent to the initial one. Clearly, the described procedure produces  $\mathcal{P}'$  in linear time.

The case when  $v_j$  is reachable from  $v_i$  in  $\mathcal{P}$  is similar. The case when neither of vertices can be reached from the other is trivial.  $\square$

(Recall that each  $\mathcal{P}$  satisfies properties in the beginning of this section.) The next algorithm makes values  $\mathcal{E}(v_i)$  and  $\mathcal{E}(v_j)$  different by doubling the value of  $\mathcal{E}(v_j)$ .  $\mathcal{E}$ -values of all other vertices remain the same. For convenience we use the following notation throughout the rest of the paper. We denote by  $v^{(n)}$  a vertex such that  $\mathcal{E}(v) = n$  (if it exists). Recall that for each vertex  $v \in \mathcal{P}$  the value  $\mathcal{E}(v)$  is a power of two. Hence if  $n$  is not a power of 2 then a vertex  $v^{(n)}$  does not exist in  $\mathcal{P}$ .

**Algorithm 5.6.** (*Double  $\mathcal{E}$ -value of a vertex*)  $\mathcal{P}' = \text{Double}(\mathcal{P}, v_i, v_j)$ .

INPUT. A circuit  $\mathcal{P}$  on vertices  $\{v_1, \dots, v_n\}$  with the specified pair of vertices  $v_i, v_j$  such that  $\mathcal{E}(v_i) = \mathcal{E}(v_j)$ .

OUTPUT. An equivalent circuit  $\mathcal{P}'$  on vertices  $\{v'_1, \dots, v'_n\}$  (with, maybe, one additional vertex  $d$ ) such that  $\mathcal{E}(v'_j) = 2\mathcal{E}(v_j)$  and  $\mathcal{E}(v'_k) = \mathcal{E}(v_k)$  for  $k \neq j$ .

COMPUTATIONS.

- A) Apply Algorithm 5.4 to vertices  $v_i, v_j$  in  $\mathcal{P}$ .
- B) Double the value  $\mathcal{E}(v_j)$  as follows:
  - 1) Compute the maximal number  $N$  such that for each  $0 \leq k < N$  there exists an edge  $v_j \xrightarrow{1} v^{(2^k)}$  in  $\mathcal{P}$ .
  - 2) If  $v^{(2^N)}$  does not exist in  $\mathcal{P}$  then (see Figure 6)
    - a) add a new vertex  $d$  into  $\mathcal{P}$ ;
    - b) connect  $d$  to vertices in  $\{v^{(2^0)}, \dots, v^{(2^{N-1})}\}$  in such a way that  $\mathcal{E}(d) = 2^N$  (by Lemma 2.1 it is possible);
    - c) remove all the edges  $v_j \xrightarrow{1} v^{(2^k)}$  (for each  $0 \leq k < N$ );

- d) add an edge  $v_j \xrightarrow{1} d$ .
- 3) The case when  $v^{(2^N)}$  exists in  $\mathcal{P}$  and there is an edge  $v_j \xrightarrow{-1} v^{(2^N)}$  is impossible since by assumption there are no superfluous edges in  $\mathcal{P}$ .
- 4) If  $v^{(2^N)}$  exists in  $\mathcal{P}$  and there is no edge between  $v_j$  and  $v^{(2^N)}$  then:
  - a) remove all the edges  $v_j \xrightarrow{1} v^{(2^k)}$  (for each  $0 \leq k < N$ );
  - b) add the edge  $v_j \xrightarrow{1} v^{(2^N)}$ .
- C) (Update edges) For each  $k \in \{1, \dots, n\} \setminus \{i, j\}$  do the following:
  - 1) If there exist edges  $e_1 = v_k \xrightarrow{\pm 1} v_i$  and  $e_2 = v_k \xrightarrow{\pm 1} v_j$  (labels are equal) then erase the edge  $e_1$ .
  - 2) If there exist edges  $e_1 = v_k \xrightarrow{\pm 1} v_i$  and  $e_2 = v_k \xrightarrow{\mp 1} v_j$  (labels are opposite) then erase both edges.
  - 3) If there exists exactly one of the edges  $e_1 = v_k \xrightarrow{l} v_i$  and  $e_2 = v_k \xrightarrow{l} v_j$  then erase it and add an edge  $e_1 = v_k \xrightarrow{l} v_i$  (with the same label).
- D) Update marks on  $v_i$  and  $v_j$ :
  - 1) If both  $v_i$  and  $v_j$  are marked and  $\nu(v_i) = \nu(v_j)$  in  $\mathcal{P}$  then unmark  $v_i$ .
  - 2) If both  $v_i$  and  $v_j$  are marked and  $\nu(v_i) = -\nu(v_j)$  in  $\mathcal{P}$  then unmark both  $v_i$  and  $v_j$ .
  - 3) If exactly one of  $v_i, v_j$  has a mark  $\nu = \pm 1$  in  $\mathcal{P}$  then unmark it and mark  $v_i$  with  $\nu$ .
- E) Output the result.

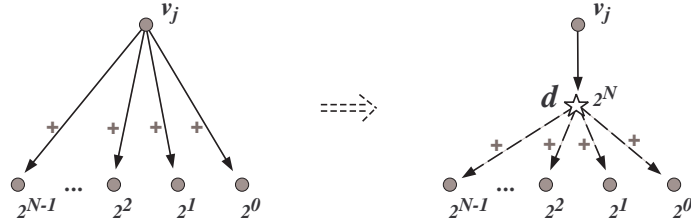


Figure 6: Introducing auxiliary vertex  $d$  (case B.2 of Algorithm 5.6). The dashed lines denote “possible” edges. Grey vertices can be marked or unmarked.

Let  $\mathcal{P}'$  be the result of an application of Algorithm 5.6 to  $\mathcal{P}$ . Observe that Algorithm 5.6 does not remove any vertices, but might introduce a new vertex into  $\mathcal{P}$  at step B.2. We will refer to this vertex as an *auxiliary* vertex and denote it by  $d$ . Furthermore, if  $V(\mathcal{P}) = \{v_1, \dots, v_n\}$  then  $V(\mathcal{P}')$  contains vertices  $\{v'_1, \dots, v'_n\}$  where each  $v'_k \in V(\mathcal{P}')$  corresponds to  $v_k \in V(\mathcal{P})$ , and perhaps a new vertex  $d$  which we call an *auxiliary vertex*.

**Proposition 5.7.** Let  $\mathcal{P}' = \text{Double}(\mathcal{P}, v_i, v_j)$ . Then  $\mathcal{P}'$  is equivalent to  $\mathcal{P}$ . Moreover,  $\mathcal{E}(v'_j) = 2\mathcal{E}(v_j)$  and  $\mathcal{E}(v'_k) = \mathcal{E}(v_k)$  for each  $k \neq j$ .

*Proof.* By definition  $\mathcal{E}(v_j) = 2^{p_j}$ , where  $p_j = \sum_{e \in \text{Out}_{v_j}} \mu(e)\mathcal{E}(\beta(e))$ . Let  $N$  be the maximal number such that for each  $k < N$ , there exists an edge  $v_j \xrightarrow{1} v^{(2^k)}$  in  $\mathcal{P}$  (computed at step B.1). Steps B.2), B.3), and B.4) are mutually exclusive. We consider only one case defined in B.2 ( $\mathcal{P}$  does not contain a vertex  $v^{(2^N)}$ ) Other cases can be considered similarly.

In the case under consideration the algorithm creates an unmarked vertex  $d$  such that  $\mathcal{E}(d) = 2^N$ , removes edges  $\{v_j \xrightarrow{1} v^{(2^0)}, \dots, v_j \xrightarrow{1} v^{(2^{N-1})}\}$  leaving  $v_j$ , and adds an edge  $v_j \xrightarrow{1} d$ . Clearly, after these transformations  $\mathcal{E}(v'_j) = 2^{p'_j}$  where

$$p'_j = p_j - (2^{N-1} + \dots + 2^0) + 2^N = p_j + 1$$

and, therefore, after the step B.2 we have  $\mathcal{E}(v'_j) = 2\mathcal{E}(v_j)$ .

Let  $v_k$  be a vertex in  $\mathcal{P}$  such that there is an edge  $v_k \xrightarrow{\pm 1} v_j$ . Since  $\mathcal{E}(v_j)$  has been changed the value  $\mathcal{E}(v_k)$  has been changed too. On step C) the algorithm recovers  $\mathcal{E}$ -values of these vertices. It is straightforward to check (using the definition of  $\mathcal{E}$ ) that after the step C) we have  $\mathcal{E}(v'_k) = \mathcal{E}(v_k)$  for all  $k \neq j$ .

Finally, at step D), Algorithm 5.6 makes sure that  $\mathcal{E}(\mathcal{P}) = \mathcal{E}(\mathcal{P}')$ . If the vertex  $v_j$  is marked then after the step D) we have  $\mathcal{E}(\mathcal{P}') = \mathcal{E}(\mathcal{P}) + \nu(v_j)\mathcal{E}(v_j)$ . To get the equality back Algorithm 5.6 performs the step D), which removes the additional summand. □

**Proposition 5.8.** The time-complexity of Algorithm 5.6 is  $O(n)$  where  $n$  is the number of vertices in  $\mathcal{P}$ .

*Proof.* Performing step A) requires  $O(n)$  operations, step B) requires  $O(n)$  operations, step C) requires  $O(n)$  operations, step D) requires  $O(1)$  operations. □

Observe, that after doubling the value  $\mathcal{E}(v_j)$  we have  $\mathcal{E}(v_j) = 2\mathcal{E}(v_i) \neq \mathcal{E}(v_i)$ . But it is possible that there is a vertex  $v'_k \in \mathcal{P}'$  such that  $\mathcal{E}(v'_j) = \mathcal{E}(v'_k)$ . In this case we have to double the value  $\mathcal{E}(v'_j)$  again. We formalize it in the following algorithm.

**Algorithm 5.9.** ( $\mathcal{E}$ -value separation)  $\mathcal{P}' = \text{Separate}(\mathcal{P}, v_j)$ .

INPUT. A circuit  $\mathcal{P}$  with equivalent vertices  $v_i$  and  $v_j$ .

OUTPUT. A reduced circuit  $\mathcal{P}'$  equivalent to  $\mathcal{P}$ .

COMPUTATIONS.

- A) Double the value  $\mathcal{E}(v_j)$  in  $\mathcal{P}$  by Algorithm 5.6. Denote the result by  $\mathcal{P}'$  (i.e.  $\mathcal{P}' \leftarrow \text{Double}(\mathcal{P}, v_j)$ ).
- B) Trim  $\mathcal{P}'$  (using Algorithm 4.3). Denote the result by  $\mathcal{P}$  (i.e.  $\mathcal{P} \leftarrow \text{Trim}(\mathcal{P}')$ ).

- C) If  $v_j$  was not removed on step B) (see Figure 7 for example) and there exists a vertex  $v_k \in \mathcal{P}$  such that  $\mathcal{E}(v_k) = \mathcal{E}(v_j)$  then goto A) and repeat for  $v_k$  and  $v_j$ .
- D) Output  $\mathcal{P}$ .

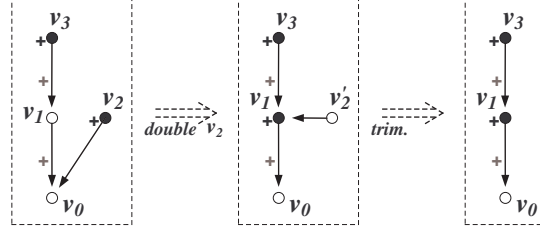


Figure 7: Doubling of  $\mathcal{E}(v_2)$  results in a non-trimmed circuit and a removal of  $v_2'$ . Only one iteration (application of Algorithm 5.6) is performed.

**Proposition 5.10.** Let  $\mathcal{P}'$  be the output of Algorithm 5.9 on a circuit  $\mathcal{P}$ . Then  $\mathcal{P}'$  is equivalent to  $\mathcal{P}$ .

*Proof.* Follows from Proposition 5.7. □

One can find an upper bound for the number of iterations Algorithm 5.9 performs to make  $\mathcal{E}(v_j)$  unique among  $\mathcal{E}$ -values of vertices in  $\mathcal{P}$ . Let

$$v_{a_1}, \dots, v_{a_s} \tag{3}$$

be a sequence of vertices in  $\mathcal{P}$  such that:

- 1)  $\mathcal{E}(v_{a_1}) = \mathcal{E}(v_j)$ , where  $v_{a_1}$  and  $v_j$  are distinct vertices;
- 2)  $\mathcal{E}(v_{a_{k+1}}) = 2\mathcal{E}(v_{a_k})$  for each  $k = 1, \dots, s-1$ ;
- 3)  $s$  is the maximal length of a sequence with such properties.

We call the sequence (3) satisfying all the properties above the *separation sequence* for  $v_j$  in  $\mathcal{P}$ . The number of iterations Algorithm 5.9 performs to separate  $v_j$  is not greater than  $s$  as shown in Figure 7.

As we mentioned earlier each application of Algorithm 5.6 can introduce one auxiliary vertex. Algorithm 5.9 invokes Algorithm 5.6 several times and, hence, several auxiliary vertices might be introduced. In the next proposition we show that an application of Algorithm 5.9 introduces at most one additional vertex.

**Proposition 5.11.** Let  $\mathcal{P}$  be a power circuit. Suppose  $v_i$  and  $v_j$  are the only two equivalent vertices in  $\mathcal{P}$ . Let  $v_{a_1}, \dots, v_{a_s}$  be a separation sequence for  $v_j$ . Then during the separation of  $\mathcal{E}(v_j)$  from  $\mathcal{E}(v_{a_1}), \dots, \mathcal{E}(v_{a_s})$  an auxiliary vertex can be introduced at the last iteration only. Therefore, Algorithm 5.9 can introduce at most one auxiliary vertex and if  $\mathcal{P}' = \text{Separate}(\mathcal{P}, v_i, v_j)$  then  $|V(\mathcal{P}')| \leq |V(\mathcal{P})| + 1$ .

*Proof.* Assume that an auxiliary vertex  $d$  was introduced on  $k$ th iteration ( $k < s$ ) (when separating  $\mathcal{E}(v_j)$  and  $\mathcal{E}(v_{a_k})$ ). Let  $\mathcal{E}(d) = 2^N$  for some  $N \in \mathbb{N}$ . Denote by  $v'_j$  the vertex  $v_j$  after the  $k$ th iteration (technically  $v_j$  and  $v'_j$  belong to different graphs). See Figure 8.

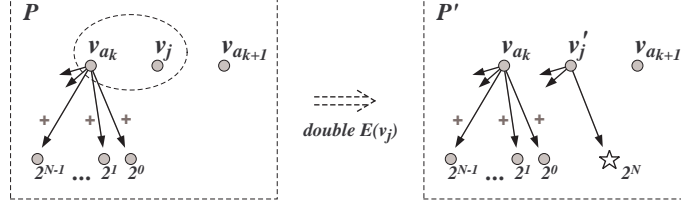


Figure 8: A situation when an auxiliary vertex  $d$  (such that  $\mathcal{E}(d) = 2^N$ ), denoted by the star, was introduced in the middle of separation. In this case we argue that the vertex  $v_{a_{k+1}}$  must be already connected to a vertex with  $\mathcal{E}$ -value equal to  $2^N$ .

Consider vertices  $v_{a_k}$  and  $v_{a_{k+1}}$  in the circuit before the  $k$ th iteration. We have  $\mathcal{E}(v_{a_k}) = \mathcal{E}(v_j) = 2^p$  and  $\mathcal{E}(v_{a_{k+1}}) = \mathcal{E}(v'_j) = 2^{p'}$ , where

$$p = \sum_{e \in \text{Out}_{v_j}} \mu(e) \mathcal{E}(\beta(e)) = \dots + (2^{N-1} + \dots + 2^0)$$

and

$$p' = \sum_{e \in \text{Out}_{v'_j}} \mu(e) \mathcal{E}(\beta(e)) = p + 1 = \dots + 2^N.$$

Observe that since vertices  $v_j, v_{a_k}$  are not reachable from each other and since  $v_j, v_{a_k}$  is the only pair of equivalent vertices it follows that the binary sum above for  $p$  is reduced. Also, by our assumption (auxiliary vertex is created after the  $k$ th separation), there is no edge  $e \in \text{Out}_{v_{a_k}}$  such that  $\mathcal{E}(\beta(e)) = 2^N$ . Therefore,  $2^N$  is the smallest summand in the binary sum for  $p'$  above and  $p'$  is divisible by  $2^N$ . Hence,  $v_{a_{k+1}}$  cannot be connected to vertices  $v^{(2^0)}, \dots, v^{(2^{N-1})}$  (otherwise it would contradict divisibility of  $p'$  by  $2^N$  or the fact that  $\mathcal{P}$  does not contain multiple edges). And there must exist a vertex  $v^{(2^N)}$  in the circuit before the  $k$ th separation and  $v_{a_{k+1}}$  must be connected to it (follows from Lemma 2.3). Obtained contradiction finishes the proof.  $\square$

For some circuits it is impossible to avoid adding a new vertex when performing separation. Figure 9 illustrates the case.

**Proposition 5.12.** The time complexity of Algorithm 5.9 is  $O(n^2)$ , where  $n$  is the number of vertices in  $\mathcal{P}$ .

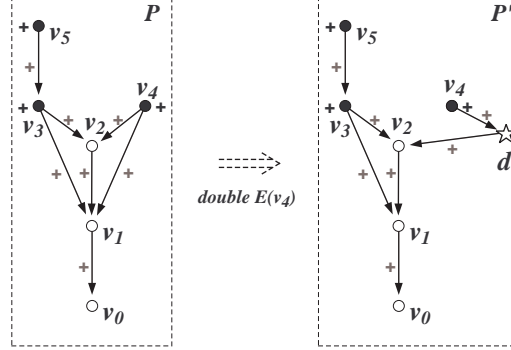


Figure 9: A situation when it is necessary to introduce an auxiliary vertex to “separate” vertices  $v_3$  and  $v_4$  ( $\mathcal{E}(v_3) = \mathcal{E}(v_4) = 8$ ). To double the value  $\mathcal{E}(v_4)$  we have to add one to the power of  $v_4$  which is  $\mathcal{E}(v_1) + \mathcal{E}(v_2) = 2 + 1$  initially. Therefore, a new vertex  $d$  such that  $\mathcal{E}(d) = 4$  is required.

*Proof.* In the worst case one has to double the value of  $v_j$  at most  $n$  times which makes the complexity of Algorithm 5.9 at most quadratic in terms of  $|V(\mathcal{P})|$ .  $\square$

Finally, notice that in Algorithm 5.6 we assume that we know the  $\mathcal{E}$ -values of vertices. In the next section we explain how it can be achieved.

### 5.3 Reduction process

In this section we present an algorithm which transforms any power circuit into an equivalent reduced one. Moreover, we show that this operation can be performed in polynomial time in terms of the size of the input.

First, we describe the idea of the algorithm. Let  $\mathcal{P}$  be a trimmed circuit without redundant zeros (described in Section 4.1). Assume that, in addition, we are provided with a subset  $C$  of  $V(\mathcal{P})$  satisfying the following properties:

- (C1) For  $u, v \in C$ ,  $\mathcal{E}(u) = \mathcal{E}(v)$  if and only if  $u = v$  (i.e., property (R2) holds inside  $C$ ).
- (C2) If  $u \in C$  and  $u \rightarrow v$  is an edge leaving  $u$  then  $v \in C$  ( $C$  is itself a circuit).

Moreover, assume that we have the following additional information about  $C$ :

- 1) Vertices from  $C$  are ordered with respect to their  $\mathcal{E}$ -values. In other words there exists a sequence  $c_1, \dots, c_m$  such that  $C = \{v_{c_1}, \dots, v_{c_m}\}$  and  $\mathcal{E}(v_{c_i}) < \mathcal{E}(v_{c_j})$  if and only if  $i < j$ .
- 2) There is a sequence  $d_1, \dots, d_{m-1}$  of 0's and 1's such that  $d_i = 1$  if and only if  $\mathcal{E}(v_{c_{i+1}}) = 2\mathcal{E}(v_{c_i})$ .



An iteration of the reduction process transforms  $\mathcal{P} = (V, E)$  and extends the set  $C$  so that the number  $|V| - |C|$  decreases by at least one. The reduction procedure works until  $C = V$ . The main ingredient is Algorithm 5.9 described in Section 5.2.

The initial set  $C$  is computed as follows. Let  $\{v_1, \dots, v_n\}$  be a geometric order on  $V(\mathcal{P})$ . If  $n = 1$  then  $\mathcal{E}(\mathcal{P}) = 0$  and  $\mathcal{P}$  is reduced. Suppose  $n \geq 2$ . It follows from Lemma 5.3 that  $\mathcal{E}(v_1) = 0$  and  $\mathcal{E}(v_2) = 1$ . Put  $C = \{v_1, v_2\}$ ,  $c_1 = 1$ ,  $c_2 = 2$ , and  $d_1 = 0$ . This is the basis of computations.

Now we describe one iteration. If  $m = n$  then  $\mathcal{P}$  is reduced and there is nothing to do. Suppose that  $m \neq n$ . Since  $\mathcal{P}$  has no loops, there exists a vertex  $v \in V \setminus C$  such that  $C' = C \cup \{v\}$  satisfies property (C2). Our main goal is to make  $C'$  satisfy property (C1). In the next lemma we show some computational properties of the set  $C'$ .

**Lemma 5.13.** *Let  $v$  be a vertex in  $V \setminus C$  such that  $C \cup \{v\}$  satisfies property (C2). For any vertex  $u \in C$  one can compare values  $\mathcal{E}(v)$  and  $\mathcal{E}(u)$  and check if  $\mathcal{E}(v) = 2\mathcal{E}(u)$  or  $\mathcal{E}(u) = 2\mathcal{E}(v)$ . Furthermore, the time complexity of this operation is  $O(|C|)$ .*

*Proof.* By definition  $\mathcal{E}(v) = 2^{p_v}$  and  $\mathcal{E}(u) = 2^{p_u}$  where

$$p_v = \sum_{e \in \text{Out}_v} \mu(e) \mathcal{E}(\beta(e)) \quad \text{and} \quad p_u = \sum_{e \in \text{Out}_u} \mu(e) \mathcal{E}(\beta(e)).$$

Clearly,  $\mathcal{E}(v) < \mathcal{E}(u)$  if and only if  $p_v < p_u$ . Hence, it is sufficient to compare  $p_v$  and  $p_u$ . It follows from the choice of  $v$  and property (C2) that edges leaving  $v$  and  $u$  have termini in  $C$  and, therefore, the binary sums above for  $p_v$  and  $p_u$  are reduced by (C1). Moreover, by assumption, vertices from  $C$  are ordered with respect to their  $\mathcal{E}$ -values and we know  $\mathcal{E}$ -values of which of them are doubles  $\mathcal{E}$ -values of others vertices (provided by the sequence  $d_1, \dots, d_{m-1}$ ). This information is clearly enough to use Algorithm 2.7 which has linear time complexity by Proposition 2.6. Since  $|\text{Out}_u| \leq |C|$  and  $|\text{Out}_v| \leq |C|$  the linearity of the process follows.

Finally, since Algorithm 2.7 can determine if  $p_u$  and  $p_v$  differ by  $\pm 1$ , one can determine whether  $\mathcal{E}(v) = 2\mathcal{E}(u)$  or  $\mathcal{E}(u) = 2\mathcal{E}(v)$ . □

Now we can describe the inductive step. By Lemma 5.13 one can compare the vertex  $v$  with any vertex  $u \in C$  and, hence, find a position of  $v$  in the ordered sequence  $\{v_{c_1}, \dots, v_{c_m}\} = C$ . (Observe that to find a position of  $v$  one does not have to compare  $v$  with each  $u \in C$ . Instead, this can be achieved by a binary search in at most  $\log_2 m$  comparisons.) There are two outcomes of the comparison of  $v$  with the vertices from  $C$  possible. First, if for each  $u \in C$   $\mathcal{E}(v) \neq \mathcal{E}(u)$  then we can add  $v$  into  $C$  without any modification of a current circuit and update the sequences  $\{c_1, \dots, c_m\}$  and  $\{d_1, \dots, d_{m-1}\}$  according to the results of comparison. After that  $V \setminus C$  becomes smaller and induction hypothesis applies.

In the second case there exists a vertex  $u \in C$  such that  $\mathcal{E}(v) = \mathcal{E}(u)$ . In this case we apply Algorithm 5.9 to  $v$  to make  $\mathcal{E}(v)$  different from values  $\{\mathcal{E}(v_{c_1}), \dots, \mathcal{E}(v_{c_m})\}$ . We would like to emphasize here that the new value  $\mathcal{E}(v)$  might be equal to  $\mathcal{E}(w)$  for some  $w \in V \setminus C$ , but it is unique in  $C \cup \{v\}$ . After that we can add  $v$  into  $C$  and update the order. Also, notice that after the separation an auxiliary vertex might appear. But since it has a unique  $\mathcal{E}$ -value (in  $C$ ) we can add it into  $C$  too. It follows that  $|V \setminus C|$  becomes smaller and induction hypothesis applies.

**Algorithm 5.14.** (*Reduction*)  $\mathcal{P}' = \text{Reduce}(\mathcal{P})$ .

INPUT. A circuit  $\mathcal{P}$

OUTPUT. A reduced circuit  $\mathcal{P}'$  equivalent to  $\mathcal{P}$ .

INITIALIZATION.  $C = \emptyset$ .

COMPUTATIONS.

- A) Let  $\mathcal{P}_1 = \text{Trim}(\mathcal{P})$ .
- B)  $\mathcal{P}_2 = \text{RemoveRedundancies}(\mathcal{P}_1)$  (Algorithm 4.3).
- C) Order vertices  $V(\mathcal{P}_2)$  with respect to the geometry of  $\mathcal{P}_2$ 

$$V(\mathcal{P}_2) = \{v_1, \dots, v_n\}.$$
- D) Put  $C = \{v_1, v_2\}$  and, accordingly, initialize sequences  $c_1, c_2$  and  $d_1$ .
- E) For each vertex  $v \in \{v_3, \dots, v_n\}$  (in the order defined by indices) perform the following operations:
  - 1) remove opposite and superfluous pairs of edges leaving  $v$ ;
  - 2) using binary search and Algorithm 2.7 find a position of  $v$  in  $C$ ;
  - 3) if necessary separate the vertex  $v$  from vertices from  $C$  using Algorithm 5.9;
  - 4) add  $v$  and, perhaps, a new auxiliary vertex  $d$  into  $C$  and update the order on  $C$ .
- F) Output the obtained circuit.

The sequence of operations E.1)-E.4) applied to  $v \in \{v_3, \dots, v_n\}$  will be referred to as processing of the vertex  $v$ . Figure 10 illustrates the execution of Algorithm 5.14 for a particular circuit.

**Proposition 5.15.** Let  $\mathcal{P}' = \text{Reduce}(\mathcal{P})$ . Then  $\mathcal{E}(\mathcal{P}) = \mathcal{E}(\mathcal{P}')$ .

*Proof.* Follows from Propositions 4.4, and 5.10. □

By Proposition 5.11 each separation might introduce at most one new auxiliary vertex. Therefore, in the worst case an application of Algorithm 5.14 to  $\mathcal{P}$  can introduce  $n - 2$  new vertices. In the next proposition we show that the number of vertices in  $\mathcal{P}$  after an application of Algorithm 5.14 can increase by at most 1.

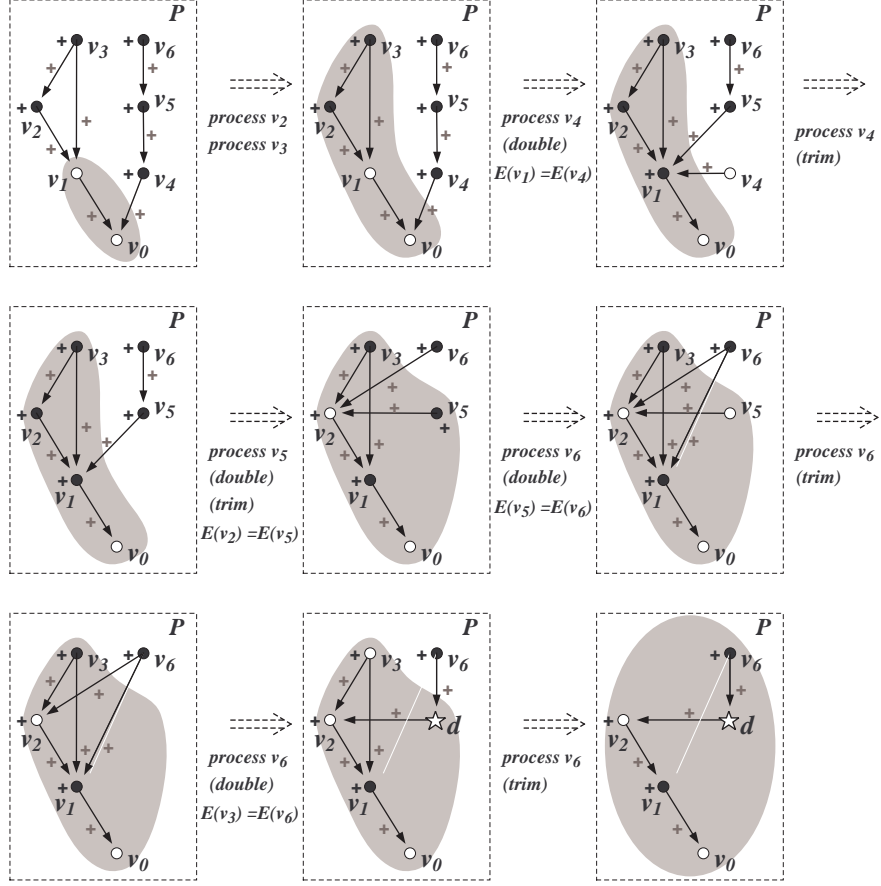


Figure 10: Reduction of a circuit. Initially  $V(\mathcal{P}) = \{v_0, \dots, v_6\}$ , where vertices are ordered in geometric order. Grey regions encompass vertices belonging to  $C$ .

**Proposition 5.16.** Let  $\mathcal{P}' = \text{Reduce}(\mathcal{P})$ . Then  $|V(\mathcal{P}')| \leq |V(\mathcal{P})| + 1$ .

*Proof.* It is convenient to introduce the following notation. Let  $v$  and  $v'$  be two vertices in  $\mathcal{P}$  such that  $\mathcal{E}(v) = 2\mathcal{E}(v')$ . In this event we say that  $v'$  is a *half* of  $v$  and denote it by  $\tilde{v}$ . Also, denote by  $\mathcal{P}_k$  a circuit obtained after processing a vertex  $v_k$  (where  $3 \leq k \leq n$ ) and by  $C_k$  the set of checked vertices in  $\mathcal{P}_k$ . For notational convenience define  $\mathcal{P}_2 = \mathcal{P}$  and  $C_2 = \{v_1, v_2\}$ . Schematically,

$$(\mathcal{P}, \{v_1, v_2\}) = (\mathcal{P}_2, C_2) \xrightarrow{\text{process } v_3} (\mathcal{P}_3, C_3) \xrightarrow{\text{process } v_4} \dots \xrightarrow{\text{process } v_n} (\mathcal{P}_n, C_n).$$

The number of vertices in  $\mathcal{P}_k$  changes at steps E.3 only when Algorithm 5.9 is used. Recall that Algorithm 5.9:

- can introduce at most one auxiliary vertex;
- remove some of the vertices while trimming the result (Algorithm 5.9 step B).

By Proposition 5.11  $|V(\mathcal{P}_{k+1})| - |V(\mathcal{P}_k)| \leq 1$  and  $|V(\mathcal{P}_{k+1})| - |V(\mathcal{P}_k)| = 1$  if and only if processing of  $v_{k+1}$  introduced a new auxiliary vertex and no other vertices were removed while trimming. Therefore, to prove the statement of the proposition it is sufficient to prove the following assertion.

**Main assertion.** Let  $s, t$  be two integers such that  $3 \leq s < t \leq n$ ,

$$|V(\mathcal{P}_{s-1})| + 1 = |V(\mathcal{P}_s)| = \dots = |V(\mathcal{P}_{t-1})|,$$

and there were no vertices removed and no auxiliary vertices introduced while processing  $v_{s+1}, \dots, v_{t-1}$ . Processing of  $v_t$  cannot introduce an auxiliary vertex.

Let  $v_{a_1}, \dots, v_{a_k} \in V(\mathcal{P}_{s-1})$  be a separation sequence for  $v_s$ . It follows from Proposition 5.11 that there are edges  $v_{a_k} \xrightarrow{1} v^{(2^0)}, \dots, v_{a_k} \xrightarrow{1} v^{(2^{N-1})}$  in  $\mathcal{P}_{s-1}$  (just before the separation of  $v_s$ ). After the processing of  $v_s$  there is an edge  $v_s \xrightarrow{1} d$  and no edges  $v_s \xrightarrow{\pm 1} v^{(2^0)}, \dots, v_s \xrightarrow{\pm 1} v^{(2^{N-1})}$  in  $\mathcal{P}_s$ . Recall that the auxiliary vertex  $d$  is created unmarked and there is only one edge incoming into  $d$  which is  $v_s \xrightarrow{1} d$ . Moreover, the following claim is true.

**Claim 1.** With our assumptions on  $v_s, \dots, v_{t-1}$  the following is true for each  $\mathcal{P}_k$  ( $s \leq k \leq t-1$ ):

- (D1) For each  $m = 0, \dots, N$  there is a vertex  $v \in C_k$  such that  $\mathcal{E}(v) = 2^m$ .
- (D2) The vertex  $d$  is unmarked in  $C_k$ .
- (D3) If there is an edge  $w \xrightarrow{\pm 1} d$  in  $\mathcal{P}_k$  then  $w \in C_k$  (i.e., only vertices from  $C_k$  can be connected to  $d$ ).

Furthermore, if the edge  $e = w \xrightarrow{\pm 1} d$  does not exist in  $\mathcal{P}_{k'}$  then it does not exist in any  $\mathcal{P}_k$ , where  $k' < k \leq t-1$ .

- (D4) Let  $w$  be a vertex connected to  $d$ . If  $\mathcal{E}(w) = 2^{p_w}$  then  $2^N$  is the smallest summand in the corresponding binary sum  $p_w = \sum_{e \in \text{Out}_w} \mu(e) \mathcal{E}(\beta(e))$ .
- (D5) The vertex  $v_{k+1} \in \mathcal{P}_k$  is not equivalent to any vertex connected to  $d$ .
- (D6) If a vertex  $v$  is connected to  $d$  then  $\tilde{v}$  is present in  $\mathcal{P}$  and at least one of  $v, \tilde{v}$  is unmarked.
- (D7) For any vertex  $w \in \mathcal{P}$  and a vertex  $v$  connected to  $d$  there exists at most one edge  $w \xrightarrow{\pm 1} v$  or  $w \xrightarrow{\pm 1} \tilde{v}$ .

*Proof.* By induction on  $k$ . Suppose  $k = s$ . Properties (D1)-(D3) are already proved in the remark preceding the claim. Since  $v_s$  is connected to  $d$  and is not connected to  $v^{(2^0)}, \dots, v^{(2^{N-1})}$  the property (D4) is established. To show (D5) consider the vertex  $v_{s+1} \in \mathcal{P}_s \setminus C_s$  and prove that  $\mathcal{E}(v_s) \neq \mathcal{E}(v_{s+1})$ . Since all vertices leaving  $v_{s+1}$  have termini in  $C_s$  and  $C_s$  is a reduced part of  $\mathcal{P}_s$  it follows that  $\mathcal{E}(v_{s+1}) = 2^{p_{v_{s+1}}}$  where  $p_{v_{s+1}} = \sum_{e \in \text{Out}_{v_{s+1}}} \mu(e) \mathcal{E}(\beta(e))$  is a reduced binary sum which does not involve  $2^N$  (by D3). As we showed above  $p_{v_s}$  is a reduced binary sum which contains  $2^N$ . Therefore, (D5) follows from Lemma 2.3. Properties (D6) and (D7) follow from the description of Algorithm 5.6.

Assume that (D1)-(D7) hold for each  $k$  such that  $s \leq k < K \leq t - 2$  and show that they hold for  $k = K$ .

**(D1)** By induction assumption vertices  $v^{(2^0)}, \dots, v^{(2^{N-1})}, v^{(2^N)}$  are present in  $\mathcal{P}_{K-1}$ . Since no vertices are removed while processing  $v_K$  the property (D1) holds for  $K$ .

**(D2)** Let  $\{v_{a_1}, \dots, v_{a_m}\} \subseteq C_{K-1}$  be a separation sequence for  $v_K$ . By induction assumption the vertex  $d$  is unmarked in  $C_{K-1}$ . Assume, to the contrary, that  $d$  is marked in  $C_K$ . Then  $d$  must belong to  $\{v_{a_1}, \dots, v_{a_m}\}$  and, hence,  $\mathcal{E}(v_K) \leq 2^N$ . We claim that in this case processing of  $v_K$  results in a removal of  $v_K$  which will contradict to the assumption of the claim (no vertices removed).

Indeed, Algorithm 5.9 consequently doubles  $\mathcal{E}(v_K)$  (using Algorithm 5.6) and trims intermediate results. Consider a step when  $\mathcal{E}(v_K) = 2^N$  and we double  $\mathcal{E}(v_K)$  to separate it from  $\mathcal{E}(d) = 2^N$ . Denote the circuit before that separation by  $\mathcal{P}'_K$  and after it by  $\mathcal{P}''_K$ . The vertex  $d$  is unmarked in  $\mathcal{P}'_K$  and marked in  $\mathcal{P}''_K$ . Therefore, the vertex  $v_K$  is unmarked in  $\mathcal{P}''_K$  since  $d$  is unmarked in  $\mathcal{P}'_K$  (follows from the description of *Double-procedure*).

Furthermore, we claim that  $v_K$  has no incoming edges in  $\mathcal{P}''_K$ . Indeed, consider two cases. Let  $w \in C_{K-1}$ . Then, initially, there is no edge  $w \rightarrow v_K$  in  $\mathcal{P}_{K-1}$  (guaranteed by property (C2) for  $C_{K-1}$ ) and, therefore, when we continuously double the value  $\mathcal{E}(v_K)$  there is no need to introduce  $w \rightarrow v_K$ . Assume  $w \notin C_{K-1}$ . Then there is no edge  $w \rightarrow d$  by (D3) for  $C_{K-1}$ . Therefore, even if the edge  $w \rightarrow v_K$  would existed, it would be removed in  $\mathcal{P}''_K$  (when separating  $v_K$  from  $d$ ).

Thus, since  $v_K$  is not marked and has no incoming edges in  $\mathcal{P}_K''$  it will be removed while trimming  $\mathcal{P}_K''$ . This contradicts to the assumption that no vertices are removed.

**(D3)** There are three cases how a vertex  $w$  can become connected to  $d$  while processing  $v_K$ :

- 1)  $d$  belongs to the separation sequence of  $v_K$  (and  $w$  is connected to  $v_K$ );
- 2)  $w = v_K$  and a vertex  $v$  connected to  $d$  belongs to the separation sequence of  $v_K$ ;
- 3)  $w = v_K$  and a vertex  $v$  for which there are edges  $v \xrightarrow{1} v^{(2^0)}, \dots, v \xrightarrow{1} v^{(2^{N-1})}$  and no edge  $v \xrightarrow{1} v^{(2^N)}$  belongs to the separation sequence of  $v_K$ .

The first case, as shown in (D2), raises a contradiction. Therefore, only the vertex  $v_K$  can become connected to  $d$ . Since it is being added to  $C_K$  it does not contradict to (D3).

Furthermore, if a vertex  $v_K$  is not connected to  $d$  in  $\mathcal{P}_K$  it is not connected to  $d$  in each  $\mathcal{P}_k$  ( $K \leq k \leq t-1$ ).

**(D4)** As shown in (D3) after the processing of the vertex  $v_K$  the set  $In_d$  can increase by at most one element  $v_K \xrightarrow{\varepsilon} d$  (i.e., processing of  $v_K$  can connect to  $d$  only the vertex  $v_K$ ). Assume that  $v_K$  is connected to  $d$  in  $\mathcal{P}_{K+1}$  and contradicts to (D4). This might happen only in the second case in the proof of (D3), i.e., some vertex  $v$  connected to  $d$  belongs to the separation sequence of  $v_K$ . Let  $v_{a_b}$  be the first vertex in the separation sequence  $\{v_{a_1}, \dots, v_{a_m}\} \subseteq C_{K-1}$  of  $v_K$  connected to  $d$ . We argue (as in the proof of (D2)) that separation of  $v_K$  results in a removal of  $v_K$  from the circuit.

The vertex  $v_K$  is not connected in  $\mathcal{P}_{K-1}$  to  $d$  by (D3). Moreover,  $v_K$  is not equivalent to any vertex in  $\mathcal{P}_{K-1}$  connected to  $d$  by (D5). Therefore,  $v_{a_b}$  is not the first vertex in the separation sequence of  $v_K$ , it must be preceded by  $\tilde{v}_{a_b}$  (which by (D6) exists in  $C_{K-1}$ ). Consider a step of doubling of  $\mathcal{E}(v_K)$  when  $\mathcal{E}(v_K) = \mathcal{E}(v_{a_b})$ . Denote by  $\mathcal{P}_K'$  the circuit before that step and by  $\mathcal{P}_K''$  the result of doubling.

The vertex  $v_K$  in  $\mathcal{P}_K''$  is unmarked since by (D6) either  $\tilde{v}_{a_b}$  or  $v_{a_b}$  is unmarked in  $\mathcal{P}_{K-1}$ . Also, by (D7) for each  $w \in \mathcal{P}_{K-1}$  there is at most one edge  $w \xrightarrow{\pm 1} \tilde{v}_{a_b}$  or  $w \xrightarrow{\pm 1} v_{a_b}$ . Therefore, in  $\mathcal{P}_K''$   $v_K$  has no incoming edges. Thus,  $v_K$  will be removed while trimming  $\mathcal{P}_K''$ . This contradicts to our assumption that no vertices are removed.

The obtained contradiction implies that a vertex connected to  $d$  cannot belong to the separation sequence of  $v_K$ . Therefore, if the vertex  $v_K$  is connected to  $d$  in  $\mathcal{P}_K$  then it cannot be connected to vertices with smaller  $\mathcal{E}$ -values ( $2^0, \dots, 2^{N-1}$ ) and, hence,  $\mathcal{E}(d) = 2^N$  is the least summand in the power of  $\mathcal{E}(v_K)$ .

**(D5)** Let  $v$  be a vertex connected to  $d$  in  $\mathcal{P}_K$ . By (D4)  $2^N$  is the least summand in the power of  $\mathcal{E}(v)$ . Hence, if  $v_{K+1}$  is equivalent to  $v$  then by Lemma 2.3  $v_{K+1}$

must be connected to a vertex with  $\mathcal{E}$ -value  $2^N$ . But termini of the edges leaving  $v_{K+1}$  belong to  $C_K$ . Therefore,  $v_{K+1}$  must be connected to  $d$  since it is the only vertex in  $C_K$  with  $\mathcal{E}$ -value  $2^N$ . Contradiction to (D3).

(D6) As shown in the proof of (D3) and (D4) a vertex can become connected to  $d$  only when its separation sequence contains a vertex  $v_{a_b}$  for which there are vertices  $v_{a_b} \xrightarrow{1} v^{(2^0)}, \dots, v_{a_b} \xrightarrow{1} v^{(2^{N-1})}$  and there is no edge  $v_{a_b} \xrightarrow{1} d$ , and  $v_{a_b}$  is the last element in the sequence. Clearly,  $v_{a_b}$  is the half of  $v_K$  in  $\mathcal{P}_K$ . It is a property of *Double*-procedure that either  $v_K$  or  $v_{a_b}$  is unmarked in  $\mathcal{P}_K$ .

(D7) Similar to the proof of (D6). □

By (D1) we have all vertices  $v^{(2^0)}, \dots, v^{(2^N)}$  in  $\mathcal{P}_{t-1}$  (where  $n \geq 2$ ). The value of a new auxiliary vertex must be strictly greater than  $2^N$  and to introduce a new auxiliary vertex we need a vertex  $v$  for which there are edges  $v \xrightarrow{1} v^{(2^0)}, \dots, v \xrightarrow{1} v^{(2^N)}$ . But by property (D5) any vertex connected to  $d = v^{(2^N)}$  has  $2^N$  as the lowest summand of its power, so it cannot be connected to the vertices  $v^{(2^0)}, \dots, v^{(2^{N-1})}$ . Thus, processing of  $v_t$  cannot introduce a new auxiliary vertex. □

The estimate  $|V(\mathcal{P}')| \leq |V(\mathcal{P})| + 1$  in the statement of Proposition 5.16 cannot be further improved. Figure 9 gives an example when  $|V(\mathcal{P}')| = |V(\mathcal{P})| + 1$ .

**Proposition 5.17.** (*Complexity of reduction*) The complexity of Algorithm 5.14 is  $O(|V(\mathcal{P})|^3)$ .

*Proof.* Denote by  $m$  the number of edges in  $\mathcal{P}$ . Observe that from property (R2) it follows that  $m \leq n^2$ .

We analyze each step in Algorithm 5.14. Trimming and removing redundancies around zero requires  $O(n + m) \leq O(n^2)$  steps. The same time complexity is required for computing the geometric order on  $\mathcal{P}$ . The most complicated part is step E). For each vertex  $v_i$ :

- 1) Removing redundancies requires at most  $O(n)$  steps.
- 2) It takes linear time  $O(n)$  to compare two  $\mathcal{E}$ -values and it will take  $O(n \log n)$  steps to find a position of  $v_i$  in the current ordered set  $C$ .
- 3) Adding  $v_i$  into  $C$  takes a constant time  $O(1)$  to perform.
- 4) Separation of  $v_i$  in  $C_i$  requires  $O(n^2)$  steps.

Therefore, processing of  $v_i$  requires in the worst case  $O(n^2)$  steps and processing of all vertices in  $\mathcal{P}$  requires  $O(n^3)$  steps. Summing all up we get the result. □



## 6 Computing normal forms

In this section we show how to find normal forms of constant power circuits.

**Lemma 6.1.** *For a constant power circuit  $\mathcal{P}$  one can check in time  $O(|V(\mathcal{P})|^3)$  whether  $\mathcal{P}$  is proper, or not.*

*Proof.* By definition  $\mathcal{P}$  is proper if and only if  $\mathcal{E}(v) \in \mathbb{N}$  for every  $v \in V(\mathcal{P})$ . This is implicitly checked in the reduction process that requires  $O(|V(\mathcal{P})|^3)$  operations.  $\square$

**Theorem 6.2.** *There exists a procedure which for any  $n \in \mathbb{N}$  computes the unique constant normal power circuit  $\mathcal{P}_n$  representing  $n$  in time  $O(\log_2 n \log_2 \log_2 n)$ . Furthermore, the circuit  $\mathcal{P}_n$  satisfies  $|V(\mathcal{P}_n)| \leq \lceil \log_2 n \rceil + 2$ .*

*Proof.* We construct a circuit for  $n$  explicitly. Put  $k = \lceil \log_2 n \rceil$  and  $V = \{0, 2^0, 2^1, \dots, 2^k\}$ . Define the set of labeled directed edges on  $V$

$$E = \{2^q \xrightarrow{\varepsilon} 2^s \mid \text{the compact sum for } q \text{ involves } \varepsilon 2^s\} \cup \{2^0 \xrightarrow{1} 0\}.$$

If  $\varepsilon_1 2^{q_1} + \dots + \varepsilon_k 2^{q_k}$  is a compact binary sum for  $n$  then put  $M = \{2^{q_1}, \dots, 2^{q_k}\}$  and  $\nu(2^{q_i}) = \varepsilon_i$ . Trim the obtained power circuit. Denote the constructed circuit by  $\mathcal{P}_n$ . It follows from construction that the obtained power circuit  $\mathcal{P}_n$  is normal and  $\mathcal{E}(\mathcal{P}_n) = n$ . Also, it follows from the construction that  $|V(\mathcal{P}_n)| \leq k + 2$  and  $|E| \leq k \log_2 k$ . Furthermore, it is straightforward to find the set  $E$ . Thus, the time complexity of the described procedure is  $O(k \log_2 k)$ .  $\square$

**Theorem 6.3.** *There exists an algorithm which for a given constant proper power circuit  $\mathcal{P}$  computes the unique (up to isomorphism) equivalent proper normal power circuit  $\mathcal{P}'$  in time  $O(|V(\mathcal{P})|^3)$ . Furthermore,  $|V(\mathcal{P}')| \leq 2|V(\mathcal{P})|$ .*

*Proof. (Step A)* Compute  $\mathcal{P}' = \text{Reduce}(\mathcal{P})$ . The reduction procedure orders the set  $V(\mathcal{P}') = \{v_1, \dots, v_n\}$  so that  $\mathcal{E}(v_i) < \mathcal{E}(v_{i+1})$ . Also, it provides us with a sequence  $d_1, \dots, d_{n-1}$  of 0's and 1's satisfying  $d_i = 1$  if and only if  $2\mathcal{E}(v_i) = \mathcal{E}(v_{i+1})$ . Since  $\mathcal{P}'$  is reduced, it follows that the sum  $\mathcal{E}(\mathcal{P}') = \sum_{v \in M} \nu(v) \mathcal{E}(v)$  is reduced and for every vertex  $v \in V(\mathcal{P}')$  the sum  $\sum_{e \in \text{Out}_v} \mu(e) \mathcal{E}(\beta(e))$  is reduced. Our goal is to make these sums compact. By Lemma 2.9 to make these sums compact we might need to introduce doubles for some vertices in  $V(\mathcal{P})$ . We do it next.

**(Step B)** Let  $\{v_{a_1}, \dots, v_{a_n}\}$  be a geometric order on  $V(\mathcal{P})$ . For every vertex  $v_{a_i}$  (from smaller indices to larger) such that  $d_{a_i} = 0$  introduce its double, i.e., add a new vertex  $v'_{a_i}$  and add edges so that  $\mathcal{E}(v'_{a_i}) = 2\mathcal{E}(v_{a_i})$  as described in Algorithm 5.6. It is very important to note that Algorithm 5.6 never performs step B.2) (and hence does not introduce new auxiliary vertices) because the vertex  $v^{2^N}$  in the description of Algorithm 5.6 is a double of some vertex  $v_{a_j}$  and it is already introduced.

**(Step C)** Next we use the procedure described in Lemma 2.8 to make sure that for every vertex  $v \in V(\mathcal{P})$  the binary sum  $\sum_{e \in \text{Out}_v} \mu(e) \mathcal{E}(\beta(e))$  is compact.

Since the doubles were introduced to  $\mathcal{P}'$  it follows from Lemma 2.9 that this can be done.

**(Step D)** To make the sum  $\mathcal{E}(\mathcal{P}) = \sum_{v \in M} \nu(v) \mathcal{E}(v)$  compact we change  $M$  and  $\nu$  as described in Lemma 2.8. By Lemma 2.9 we can do that.

**(Step E)** Finally, we trim the obtained power circuit and output the result.

The reduction step is the most time consuming step which requires  $O(|V(\mathcal{P})|^3)$  steps. Hence the claimed bound on time complexity.  $\square$

## 7 Elementary operations over power circuits

In this section we show how to efficiently perform arithmetic operations over power circuits.

### 7.1 Addition and subtraction

Let  $\mathcal{P}_1$  and  $\mathcal{P}_2$  be two circuits. The following algorithm computes a circuit  $\mathcal{P}_+$  such that  $\mathcal{T}_{\mathcal{P}_+} = \mathcal{T}_{\mathcal{P}_1} + \mathcal{T}_{\mathcal{P}_2}$  over  $\mathbb{Z}$  (or  $\mathbb{R}$ ).

**Algorithm 7.1.** (*Sum of circuits*)

INPUT. Circuits  $\mathcal{P}_1 = (\mathcal{P}_1, M_1, \mu_1, \nu_1)$  and  $\mathcal{P}_2 = (\mathcal{P}_2, M_2, \mu_2, \nu_2)$ .

OUTPUT. Circuit  $\mathcal{P}_+ = (\mathcal{P}_+, M, \mu, \nu)$  such that  $\mathcal{T}_{\mathcal{P}_+} = \mathcal{T}_{\mathcal{P}_1} + \mathcal{T}_{\mathcal{P}_2}$  over  $\mathbb{Z}$ .

COMPUTATIONS.

- A) Let  $\mathcal{P}_+$  be a disjoint union of graphs  $\mathcal{P}_1$  and  $\mathcal{P}_2$ .
- B) Put  $M = M_1 \cup M_2$ .
- C) Define a function  $\nu$  on  $M$  such that  $\nu|_{M_1} = \nu_1$  and  $\nu|_{M_2} = \nu_2$ .
- D) Define a function  $\mu$  on  $E(\mathcal{P}_+)$  such that  $\mu|_{E(\mathcal{P}_1)} = \mu_1$  and  $\mu|_{E(\mathcal{P}_2)} = \mu_2$ .
- E) Return  $\mathcal{P}_+ = (\mathcal{P}_+, M, \mu, \nu)$ .

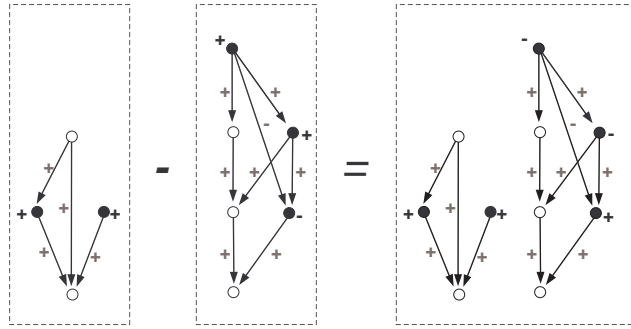


Figure 11: Difference of circuits.

**Proposition 7.2.** Let  $\mathcal{P}_1$  and  $\mathcal{P}_2$  be power circuits. Then

- 1)  $\mathcal{T}_{\mathcal{P}_+} = \mathcal{T}_{\mathcal{P}_1} + \mathcal{T}_{\mathcal{P}_2}$  over  $\mathbb{Z}$ ,
- 2) Algorithm 7.1 computes  $\mathcal{P}_+$  in linear time  $O(|\mathcal{P}_1| + |\mathcal{P}_2|)$ .
- 3) Moreover, the size of  $\mathcal{P}_+ = (\mathcal{P}_+, M, \mu, \nu)$  is bounded as follows:
  - $|V(\mathcal{P}_+)| = |V(\mathcal{P}_1)| + |V(\mathcal{P}_2)|$ ,
  - $|E(\mathcal{P}_+)| = |E(\mathcal{P}_1)| + |E(\mathcal{P}_2)|$ ,
  - $|M| = |M_1| + |M_2|$ .

*Proof.* Straightforward from the construction of  $\mathcal{E}(\mathcal{P}_+)$  in Algorithm 7.1.  $\square$

A similar result holds for subtraction  $-$ . To compute  $\mathcal{P}_- = \mathcal{P}_1 - \mathcal{P}_2$  one can modify Algorithm 7.1 as follows. At step C) instead of putting  $\nu|_{M_2} = \nu_2$  put  $\nu|_{M_2} = -\nu_2$ . Clearly, for the obtained circuit  $\mathcal{P}_-$  the equality  $\mathcal{T}_{\mathcal{P}_-} = \mathcal{T}_{\mathcal{P}_1} - \mathcal{T}_{\mathcal{P}_2}$  over  $\mathbb{Z}$ , as wells as the complexity and size estimates of Lemma 7.2 hold.

Sometimes we refer to the circuits  $\mathcal{P}_+$  and  $\mathcal{P}_-$  as  $\mathcal{P}_1 + \mathcal{P}_2$  and  $\mathcal{P}_1 - \mathcal{P}_2$ , correspondingly.

## 7.2 Exponentiation

Let  $\mathcal{P}$  be a power circuit. The next algorithm produces a circuit  $\mathcal{P}'$  such that  $\mathcal{T}_{\mathcal{P}'} = 2^{\mathcal{T}_{\mathcal{P}}}$ .

**Algorithm 7.3.** (*Exponentiation in base 2*)

INPUT. A circuit  $\mathcal{P} = (\mathcal{P}, M, \mu, \nu)$ .

OUTPUT. A circuit  $\mathcal{P}' = (\mathcal{P}', M', \mu', \nu')$  such that  $\mathcal{T}_{\mathcal{P}'} = 2^{\mathcal{T}_{\mathcal{P}}}$ .

COMPUTATIONS:

- 1) Construct a graph  $\mathcal{P}'$  as follows:
  - Add a new unmarked vertex  $v_0$  into the graph  $\mathcal{P}$ .
  - For each  $u \in M$  add an edge  $e = (v_0 \rightarrow u)$ .
- 2) Put  $M = \{v_0\}$ .
- 3) Define  $\nu'(v_0) = 1$ .
- 4) Extend  $\mu$  to  $\mu'$  defining  $\mu'$  on new edges  $(v_0, u)$  by  $\mu'(v_0, u) = \nu(u)$ .
- 5) Output  $(\mathcal{P}', M, \mu', \nu')$ .

See Figure 12 for an example.

**Proposition 7.4.** Let  $\mathcal{P}$  be a power circuit. Then

- 1)  $\mathcal{T}_{\mathcal{P}'} = 2^{\mathcal{T}_{\mathcal{P}}}$ ,
- 2) Algorithm 7.3 computes  $\mathcal{P}'$  in linear time  $O(|\mathcal{P}|)$ .

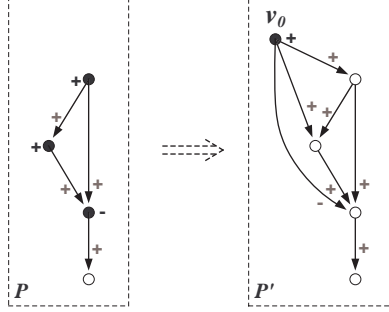


Figure 12: Exponentiation in base 2.

3) Moreover, the size of  $\mathcal{P}' = (\mathcal{P}', M', \mu', \nu')$  is bounded as follows:

- $|V(\mathcal{P}')| = |V(\mathcal{P})| + 1$ ,
- $|E(\mathcal{P}')| \leq |E(\mathcal{P})| + |V(\mathcal{P})|$ ,
- $|M'| = 1$ .

*Proof.* Recall that  $\mathcal{T}_{\mathcal{P}} = \sum_{v \in M} \nu(v)t_v$ . Therefore,  $2^{\mathcal{T}_{\mathcal{P}}} = 2^{\sum_{v \in M} \nu(v)t_v}$  which is exactly the term  $\mathcal{T}_{\mathcal{P}'}$ . The other statements follow from the constructions in Algorithm 7.3. □

Sometimes we refer to the circuit  $\mathcal{P}'$  as  $2^{\mathcal{P}}$ .

### 7.3 Multiplication

Let  $\mathcal{P}_1$  and  $\mathcal{P}_2$  be two power circuits. In this section we construct a power circuit  $\mathcal{P}_*$  such that  $\mathcal{T}_{\mathcal{P}_*} = \mathcal{T}_{\mathcal{P}_1} \cdot \mathcal{T}_{\mathcal{P}_2}$ .

**Algorithm 7.5.** (*Product of circuits*)

INPUT. Circuits  $\mathcal{P}_1$  and  $\mathcal{P}_2$ .

OUTPUT. A circuit  $\mathcal{P}_*$  such that  $\mathcal{T}_{\mathcal{P}_*} = \mathcal{T}_{\mathcal{P}_1} \cdot \mathcal{T}_{\mathcal{P}_2}$  in any exponential ring  $R$ .

COMPUTATIONS.

- A) Apply Algorithm 4.5 to get power circuits  $\mathcal{P}'_1$  and  $\mathcal{P}'_2$ , which are equivalent to  $\mathcal{P}_1$  and  $\mathcal{P}_2$  and where all marked vertices are sources.
- B) Construct  $\mathcal{P} = (V(\mathcal{P}), E(\mathcal{P}))$ , where

$$V(\mathcal{P}) = (V(\mathcal{P}'_1) \setminus M(\mathcal{P}'_1)) \cup (V(\mathcal{P}'_2) \setminus M(\mathcal{P}'_2)) \cup M(\mathcal{P}'_1) \times M(\mathcal{P}'_2).$$

and  $E(\mathcal{P})$  contains edges of three types:

- 1) for each edge  $v_1 \xrightarrow{x} v_2$  in  $\mathcal{P}'_i$  such that  $v_1, v_2 \in V(\mathcal{P}'_i) \setminus M(\mathcal{P}'_i)$  ( $i = 1, 2$ ) add an edge  $v_1 \xrightarrow{x} v_2$  into  $\mathcal{P}$ ;

- 2) for each edge  $v_1 \xrightarrow{x} v_2$  in  $\mathcal{P}'_1$ , where  $v_1$  is marked and  $v_2$  is not, and for each vertex  $v_3 \in M(\mathcal{P}'_2)$  add an edge  $(v_1, v_3) \xrightarrow{x} v_2$  into  $\mathcal{P}$ ;
  - 3) for each edge  $v_1 \xrightarrow{x} v_2$  in  $\mathcal{P}'_2$ , where  $v_1$  is marked and  $v_2$  is not, and for each vertex  $v_3 \in M(\mathcal{P}'_1)$  add an edge  $(v_3, v_1) \xrightarrow{x} v_2$  into  $\mathcal{P}$ .
- C) Put  $M = M(\mathcal{P}'_1) \times M(\mathcal{P}'_2)$  and for each  $v = (v_1, v_2) \in M$  put  $\nu(v) = \nu(v_1)\nu(v_2)$ .
- D) Output the obtained circuit  $(\mathcal{P}, M, \mu, \nu)$ .

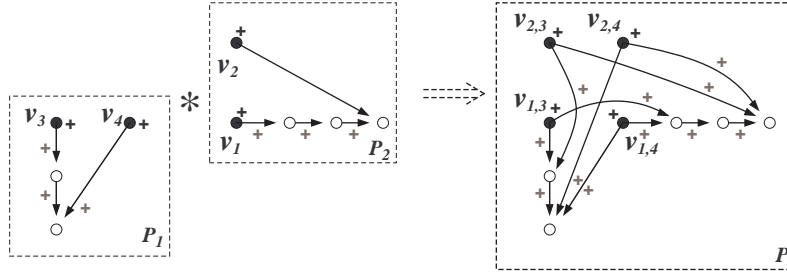


Figure 13: Circuit multiplication.

**Proposition 7.6.** Let  $\mathcal{P}_1$  and  $\mathcal{P}_2$  be two power circuits and  $\mathcal{P}_*$  obtained from them by Algorithm 7.5. Then:

- $\mathcal{T}_{\mathcal{P}_*} = \mathcal{T}_{\mathcal{P}_1} \cdot \mathcal{T}_{\mathcal{P}_2}$ ,
  - $|V(\mathcal{P}_*)| \leq |V(\mathcal{P}_1)| + |V(\mathcal{P}_2)| + |M(\mathcal{P}_1)| \cdot |M(\mathcal{P}_2)|$
- and
- $$|E(\mathcal{P}_*)| \leq 2(|V(\mathcal{P}_1)| + |V(\mathcal{P}_2)|) \cdot |V(\mathcal{P}_1)| \cdot |V(\mathcal{P}_2)|.$$
- Algorithm 7.5 computes  $\mathcal{P}_*$  in at most cubic time  $O(|V(\mathcal{P}_1)| \cdot |V(\mathcal{P}_2)| \cdot (|V(\mathcal{P}_1)| + |V(\mathcal{P}_2)|))$ .

*Proof.* Since

$$\mathcal{T}(\mathcal{P}_1) = \sum_{v_i \in M_1} \nu(v_i) 2^{\left(\sum_{e \in \text{Out}_{v_i}} \mu(e) t_{\beta(e)}\right)}$$

and

$$\mathcal{T}(\mathcal{P}_2) = \sum_{v_j \in M_2} \nu(v_j) 2^{\left(\sum_{e \in \text{Out}_{v_j}} \mu(e) t_{\beta(e)}\right)},$$

we get

$$\mathcal{T}(\mathcal{P}_1)\mathcal{T}(\mathcal{P}_2) = \sum_{v_i \in M_1, v_j \in M_2} \nu(v_i)\nu(v_j) 2^{\left(\sum_{e \in \text{Out}_{v_i}} \mu(e) t_{\beta(e)} + \sum_{e \in \text{Out}_{v_j}} \mu(e) t_{\beta(e)}\right)}$$

$$= \sum_{(v_i, v_j) \in M = M_1 \times M_2} \nu((v_i, v_j)) 2^{\sum_{e \in \text{Out}(v_i, v_j)} \mu(e) t_{\beta(e)}} = \mathcal{T}(\mathcal{P}_*).$$

To show that estimates for  $V(\mathcal{P}_*)$  and the time-complexity hold we analyze Algorithm 7.5 step by step. By Lemma 4.6 Algorithm 4.5 is linear time and the following estimates of the sizes hold:

$$|V(\mathcal{P}'_i)| \leq |V(\mathcal{P}_i)| + |M(\mathcal{P}_i)| \quad \text{and} \quad |M(\mathcal{P}'_i)| = |M(\mathcal{P}_i)|$$

(where  $i = 1, 2$ ). Hence, the time complexity of this step is at most  $O(|M(\mathcal{P}_1)| + |V(\mathcal{P}_1)| + |M(\mathcal{P}_2)| + |V(\mathcal{P}_2)|)$ . On the next two steps (B and C) we construct the graph  $\mathcal{P}$  in a very straightforward way, so the complexity of these steps is the size of  $\mathcal{P}$ . By construction of  $\mathcal{P}$  we have

$$V(\mathcal{P}_*) = (V(\mathcal{P}'_1) \setminus M(\mathcal{P}'_1)) \cup (V(\mathcal{P}'_2) \setminus M(\mathcal{P}'_2)) \cup M(\mathcal{P}'_1) \times M(\mathcal{P}'_2)$$

and the claimed estimate on  $|V(\mathcal{P})|$  holds. Clearly,  $|E(\mathcal{P})'| \leq |E(\mathcal{P}_1)| + |E(\mathcal{P}_2)| + |M(\mathcal{P}_1)| \cdot |M(\mathcal{P}_2)| \cdot (|V(\mathcal{P}_1)| + |V(\mathcal{P}_2)|) \leq 2 \cdot |V(\mathcal{P}_1)| \cdot |V(\mathcal{P}_2)| \cdot (|V(\mathcal{P}_1)| + |V(\mathcal{P}_2)|)$ . This gives the claimed upper bound on the time complexity of Algorithm 7.5.  $\square$

Sometimes we denote the circuit  $\mathcal{P}_*$  constructed above by  $\mathcal{P}_1 * \mathcal{P}_2$ .

## 7.4 Multiplication and division by a power of two

Let  $\mathcal{P}_1$  and  $\mathcal{P}_2$  be power circuits. In this section we present a procedure for constructing circuits  $\mathcal{P}_\bullet$  and  $\mathcal{P}_\circ$  such that

$$\mathcal{T}(\mathcal{P}_\bullet) = \mathcal{T}(\mathcal{P}_1) \cdot 2^{\mathcal{T}(\mathcal{P}_2)} \quad \text{and} \quad \mathcal{T}(\mathcal{P}_\circ) = \mathcal{T}(\mathcal{P}_1) \cdot 2^{-\mathcal{T}(\mathcal{P}_2)}.$$

Observe that both  $\mathcal{P}_\bullet$  and  $\mathcal{P}_\circ$  can be constructed using operations above. However, we present different more efficient procedures to build the required circuits.

**Algorithm 7.7.** (*Multiplication by a power of 2*)

INPUT. Circuits  $\mathcal{P}_1$  and  $\mathcal{P}_2$ .

OUTPUT. A circuit  $\mathcal{P}_\bullet$  such that  $\mathcal{T}(\mathcal{P}_\bullet) = \mathcal{T}(\mathcal{P}_1) \cdot 2^{\mathcal{T}(\mathcal{P}_2)}$ .

COMPUTATIONS.

- A) Construct the circuit  $\mathcal{P}'_1$  which is equivalent to  $\mathcal{P}$  and where all marked vertices are sources. Assume that  $\mathcal{P}'_1 = (\mathcal{P}'_1, M'_1, \mu'_1, \nu'_1)$  and  $\mathcal{P}_2 = (\mathcal{P}_2, M_2, \mu_2, \nu_2)$ .
- B) Define  $\mathcal{P}_\bullet = (\mathcal{P}_\bullet, M, \mu, \nu)$  as follows:
  - 1)  $\mathcal{P}_\bullet$  is a disjoint union of  $\mathcal{P}'_1$  and  $\mathcal{P}_2$ .
  - 2) For each  $v_1 \in M_1$  and each  $v_2 \in M_2$  add an edge  $v_1 \xrightarrow{\nu(v_2)} v_2$  into  $\mathcal{P}_\bullet$ .
  - 3)  $M = M_1$  and  $\nu = \nu_1$ .
- C) Output  $\mathcal{P}_\bullet$ .

Of course, the operation  $x \cdot 2^{-y}$  can be expressed via subtraction and  $x \cdot 2^y$ . However, we need a proper power circuit representation of an integer  $x \cdot 2^{-y}$ .

**Algorithm 7.8.** (*Division by a power of 2*)

INPUT. Constant power circuits  $\mathcal{P}_1$  and  $\mathcal{P}_2$ .

OUTPUT. A constant circuit  $\mathcal{P}_\circ$  such that  $\mathcal{E}(\mathcal{P}_\circ) = \mathcal{E}(\mathcal{P}_1)2^{-\mathcal{E}(\mathcal{P}_2)}$  and this is proper.

COMPUTATIONS.

- A) Let  $\mathcal{P}'_1$  be a reduced constant power circuit equivalent to  $\mathcal{P}_1$  where all marked vertices are sources. Assume that  $\mathcal{P}'_1 = (\mathcal{P}'_1, M'_1, \mu'_1, \nu'_1)$  and  $\mathcal{P}_2 = (\mathcal{P}_2, M_2, \mu_2, \nu_2)$ .
- B) Define  $\mathcal{P}_\circ$  to be  $(\mathcal{P}_\circ, M, \mu, \nu)$  where:
  - 1)  $\mathcal{P}_\circ$  is a disjoint union of  $\mathcal{P}'_1$  and  $\mathcal{P}_2$ .
  - 2) For each  $v_1 \in M_1$  and each  $v_2 \in M_2$  add an edge  $v_1 \xrightarrow{-\nu(v_2)} v_2$  into  $\mathcal{P}_\circ$ .
  - 3)  $M = M_1$  and  $\nu = \nu_1$ .
  - 4) Collapse zero vertices in  $\mathcal{P}_\circ$  (there are at least 2 of them, one coming from  $\mathcal{P}_1$  and the other from  $\mathcal{P}_2$ ).
- C) Output  $\mathcal{P}_\circ$ .

**Proposition 7.9.** Let  $\mathcal{P}_1 = (\mathcal{P}_1, M_1, \mu_1, \nu_1)$  and  $\mathcal{P}_2 = (\mathcal{P}_2, M_2, \mu_2, \nu_2)$  be circuits,  $\mathcal{P}_\bullet = \mathcal{P}_1 \bullet \mathcal{P}_2$ , and  $\mathcal{P}_\circ = \mathcal{P}_1 \circ \mathcal{P}_2$ . Then

- 1)  $\mathcal{E}(\mathcal{P}_\bullet) = \mathcal{E}(\mathcal{P}_1)2^{\mathcal{E}(\mathcal{P}_2)}$  and  $\mathcal{E}(\mathcal{P}_\circ) = \frac{\mathcal{E}(\mathcal{P}_1)}{2^{\mathcal{E}(\mathcal{P}_2)}}$ ;
- 2)  $|V(\mathcal{P}_\bullet)|, |V(\mathcal{P}_\circ)| \leq |V(\mathcal{P}_1)| + |V(\mathcal{P}_2)| + |M_1|$ .
- 3) The time complexity of Algorithm 7.7 is bounded from above by  $O(|\mathcal{P}_1| + |\mathcal{P}_2| + |M_1| \cdot |M_2|)$ .
- 4) The time complexity of Algorithm 7.8 is bounded from above by  $O(|V(\mathcal{P}_1)|^3 + |\mathcal{P}_2| + |M_1| \cdot |M_2|)$ .

*Proof.* Straightforward to check.  $\square$

We already pointed out that the operation  $\mathcal{P}_1 \circ \mathcal{P}_2$  is not defined for all pairs of power circuits  $\mathcal{P}_1 = (\mathcal{P}_1, \mu_1, M_1, \nu_1)$ ,  $\mathcal{P}_2 = (\mathcal{P}_2, \mu_2, M_2, \nu_2)$  because the value  $\mathcal{E}(\mathcal{P}_1) \cdot 2^{-\mathcal{E}(\mathcal{P}_2)}$  is not always an integer. We can naturally extend the domain of definition of  $\circ$  to the set of all pairs  $\mathcal{P}_1, \mathcal{P}_2$  by rounding the value of  $\mathcal{E}(\mathcal{P}_1) \cdot 2^{-\mathcal{E}(\mathcal{P}_2)}$ .

Our algorithms do not become less efficient if we use  $\circ$  with rounding. Indeed, if

$$\mathcal{E}(\mathcal{P}_1) = \sum_{v \in M_1} \nu_1(v) \mathcal{E}(v) \quad \text{where} \quad \mathcal{E}(v) = 2^{(\sum_{e \in \text{Out}_v} \mu_1(e) \mathcal{E}(\beta(e)))}$$

then

$$\mathcal{E}(\mathcal{P}_1) \cdot 2^{-\mathcal{E}(\mathcal{P}_2)} = \sum_{v \in M_1} \nu_1(v) 2^{\left(\sum_{e \in \text{Out}_v} \mu_1(e) \mathcal{E}(\beta(e))\right) - \mathcal{E}(\mathcal{P}_2)}.$$

To round up the value of  $\mathcal{E}(\mathcal{P}_1) \cdot 2^{-\mathcal{E}(\mathcal{P}_2)}$  it is sufficient to remove all vertices  $v$  from  $M_1$  such that  $\sum_{e \in \text{Out}_v} \mu_1(e) \mathcal{E}(\beta(e)) < \mathcal{E}(\mathcal{P}_2)$ . That can be done in polynomial time by Proposition 5.17.

## 7.5 Ordering

Clearly,  $\mathcal{E}(\mathcal{P}_1) < \mathcal{E}(\mathcal{P}_2)$  if and only if  $\mathcal{E}(\mathcal{P}_1 - \mathcal{P}_2) < 0$ . Therefore, to compare values of constant power circuits  $\mathcal{P}_1$  and  $\mathcal{P}_2$  it is sufficient to compare a value of a circuit  $\mathcal{P}_1 - \mathcal{P}_2$  with 0. For a constant power circuit  $\mathcal{P}$  define

$$\text{Sign}(\mathcal{P}) = \begin{cases} -1, & \text{if } \mathcal{E}(\mathcal{P}) < 0; \\ 0, & \text{if } \mathcal{E}(\mathcal{P}) = 0; \\ 1, & \text{if } \mathcal{E}(\mathcal{P}) > 0. \end{cases}$$

**Algorithm 7.10.** (*Sign of  $\mathcal{E}(\mathcal{P})$* )

INPUT. A circuit  $\mathcal{P}$ .

OUTPUT.  $\text{Sign}(\mathcal{P})$ .

COMPUTATIONS.

- A) Let  $\mathcal{P}' = \text{Reduce}(\mathcal{P})$  and  $C = \{v_1, \dots, v_k\}$  be a sequence of vertices produced by Algorithm 5.14 such that  $\mathcal{E}(v_i) < \mathcal{E}(v_j)$  whenever  $1 \leq i < j \leq k$ .
- B) If  $\mathcal{P}'$  is trivial then output 0.
- C) Find the marked vertex  $v_i$  in  $\mathcal{P}'$  with the greatest index  $i$ .
- D) Output  $\nu(v_i)$ .

**Proposition 7.11.** Let  $\mathcal{P}$  be a constant power circuit. Then Algorithm 7.10 computes  $\text{Sign}(\mathcal{P})$  in time bounded from above by  $O(|V(\mathcal{P})|^3)$ .

*Proof.* Let  $\mathcal{P}'$  be the reduced power circuit equivalent to  $\mathcal{P}$  produced by Algorithm 5.14, and  $C = \{v_1, \dots, v_k\}$  be a sequence of vertices produced by Algorithm 5.14 such that  $\mathcal{E}(v_i) < \mathcal{E}(v_j)$  whenever  $1 \leq i < j \leq k$ . Then

$$\mathcal{E}(\mathcal{P}) = \mathcal{E}(\mathcal{P}') = \sum_{v_j \in M} \nu(v_j) \mathcal{E}(v_j) = \sum_{v_j \in M} \nu(v_j) 2^{\left(\sum_{e \in \text{Out}_{v_j}} \mu(e) \mathcal{E}(\beta(e))\right)}$$

which is a reduced binary sum (see Section 2.1). By Proposition 4.8  $\text{Sign}(\mathcal{E}(\mathcal{P})')$  is the coefficient of the greatest power of 2, which is  $\nu(v_i)$ , where  $i$  is the greatest index such that  $v_i \in M$ . Hence  $\text{Sign}(\mathcal{E}(\mathcal{P})) = \nu(v_i)$  as claimed.

By Proposition 5.17, the reduction process performed by algorithm 5.14 has time-complexity  $O(|V(\mathcal{P})|^3)$ . Once  $\mathcal{P}$  is reduced, it is immediate to find the value of  $\nu(v_i)$ . Thus,  $O(|V(\mathcal{P})|^3)$  is an upper bound for time-complexity of Algorithm 7.10.  $\square$



## 8 Exponential algebra on power circuits

Fix a language

$$\mathcal{L} = \{+, -, *, x \cdot 2^y, x \cdot 2^{-y}, \leq, 0, 1\},$$

its sublanguage  $\mathcal{L}_0$ , which is obtained from  $\mathcal{L}$  by removing the multiplication  $*$ ; and structures

$$\mathbb{Z}_{\mathcal{L}} = \langle \mathbb{Z}; +, -, *, x \cdot 2^y, x \cdot 2^{-y}, \leq, 1 \rangle$$

and

$$\tilde{Z} = \mathbb{Z}_{\mathcal{L}_0} = \langle \mathbb{Z}; +, -, x \cdot 2^y, x \cdot 2^{-y}, \leq, 1 \rangle.$$

In this section we show that there exists an algorithm that for every algebraic  $L$ -circuit  $C$  finds an equivalent standard power circuit  $\mathcal{P}$ , or equivalently, there exists an algorithm which for every term  $t$  in the language  $\mathcal{L}$  finds a power circuit  $C_t$  which represents a term equivalent to the term  $t$  in  $\mathbb{Z}_{\mathcal{L}}$ . Moreover, if the term  $t$  is in the language  $\mathcal{L}_0$  then the algorithm computes the circuit  $C_t$  in linear time in the size of  $t$ . For integers and closed terms in  $\mathcal{L}_0$  one can get much stronger results. Let  $\mathcal{C}_{norm}$  be the set of all constant normal power circuits (up to isomorphism). We show that if  $t(X)$  is a term in  $\mathcal{L}_0$  and  $\eta : X \rightarrow \mathbb{Z}$  an assignment of variables, then there exists an algorithm which determines if  $t(\eta(X))$  is defined in  $\mathbb{Z}_{\mathcal{L}}$  (or  $\tilde{Z}$ ) or not; and if defined it then produces the normal circuit  $\mathcal{P}_t$  that presents the number  $t(\eta(X))$  in polynomial time. At the end of the section we prove that the quantifier-free theory of the structure  $\tilde{Z}$  with all the constants from  $\mathbb{Z}$  in the language is decidable in polynomial time.

### 8.1 Algebra of power circuits

We have mentioned in Introduction that every term  $t$  in the language  $\mathcal{L}$  can be realized in  $\tilde{Z}$  by an algebraic  $\mathcal{L}$ -circuit. In this section we show that every such term  $t$  also can be realized in  $\tilde{Z}$  by a power circuit  $\mathcal{P}_t$ . Furthermore, we show that if  $t$  does not involve multiplication, then the circuit  $\mathcal{P}_t$  can be computed in polynomial time in the size of  $t$  (which may not be true if  $t$  involves multiplications).

Let  $\mathcal{C}$  be the set of all power circuits in variables from a set  $X = \{x_1, x_2, \dots\}$ . Recall, that two circuits  $\mathcal{P}_1, \mathcal{P}_2 \in \mathcal{C}$  are equivalent ( $\mathcal{P}_1 \sim \mathcal{P}_2$ ) if the terms  $\mathcal{T}_{\mathcal{P}_1}$  and  $\mathcal{T}_{\mathcal{P}_2}$  define the same function in  $\tilde{Z}$ . In Section 7 we defined operations  $+, -, *, x \cdot 2^y, x \cdot 2^{-y}$  on power circuits. It is easy to see from the construction that these operations are compatible with the equivalence relation  $\sim$ , so they induce the corresponding operations on the quotient set  $\mathcal{C}/\sim$ , forming an algebraic  $\mathcal{L}$ -structure

$$\mathcal{C} = \langle \mathcal{C}/\sim; +, -, *, x \cdot 2^y, x \cdot 2^{-y}, 0, 1 \rangle$$

where we interpret the constants 0,1 by the equivalent classes of the normal power circuits with the values 0 and 1.

To clarify the algebraic structure of  $\mathcal{C}$  we need the following. Let  $\mathcal{T}_{\mathcal{L}}$  be the set of all terms in the language  $\mathcal{L}$ . Two terms  $t_1$  and  $t_2$  are termed *equivalent*

( $t_1 \sim t_2$ ) if they define the same functions on  $\tilde{Z}$ . The quotient set  $\mathcal{T}_{\mathcal{L}} / \sim$  can be naturally identified with the set  $\mathcal{F}_{\mathcal{L}}$  of all *term functions* induced by terms from  $\mathcal{T}_{\mathcal{L}}$  in  $\tilde{Z}$ . Obviously, the operations in  $\mathcal{C}$  are precisely the same as the corresponding operations over the term functions in  $\mathcal{F}_{\mathcal{L}}$ .

Denote by  $C_0, C_1$  and  $C_x$  some standard circuits that realize the terms 0, 1 and a variable  $x \in X$ . Define a map

$$\tau : \mathcal{T}_{\mathcal{L}} \rightarrow \mathcal{C}$$

by induction on complexity of the terms:

- if  $t \in \{0, 1\} \cup X$  then  $\tau(t) = C_t$ ;
- if  $t = f(t_1, t_2)$  where  $t_1, t_2$  are terms and  $f$  is an operation from  $\mathcal{L}$  then the circuit  $\tau(t) = f(\tau(t_1), \tau(t_2))$  is obtained from  $\tau(t_1)$  and  $\tau(t_2)$  as described in Section 7.

The next proposition immediately follows from the construction.

**Proposition 8.1.** The following hold:

- (1)  $\tau$  induces an isomorphism of the algebraic structures

$$\tau : \langle \mathcal{T}_{\mathcal{L}} / \sim; +, -, *, x \cdot 2^y, x \cdot 2^{-y}, 0, 1 \rangle \rightarrow \langle \mathcal{C} / \sim; +, -, *, x \cdot 2^y, x \cdot 2^{-y}, 0, 1 \rangle.$$

- (2) Let  $t \in \mathcal{T}_{\mathcal{L}}$  and  $\mathcal{P} = \tau(t)$ . Then the terms  $t$  and  $\mathcal{P}$  are equivalent in  $\mathbb{Z}_{\mathcal{L}}$ .

**Corollary 8.2.** There is an algorithm that for every algebraic  $L$ -circuit  $C$  finds an equivalent standard power circuit  $\mathcal{P}$ .

Let  $\mathcal{T}_{\mathcal{L}_0}$  be a subset of  $\mathcal{T}_{\mathcal{L}}$  consisting of terms in the language  $\mathcal{L}_0$ . We prove now that the restriction of  $\tau$  on  $\mathcal{T}_{\mathcal{L}_0}$  is linear time computable in the size of an input term  $t$  (the number  $|t|$  of operations that occur in  $t$ ).

**Theorem 8.3.** *Given  $t \in \mathcal{T}_{\mathcal{L}_0}$  it requires at most  $O(|t|)$  steps to compute  $\mathcal{P} = \tau(t)$ . Furthermore,  $|M(\mathcal{P})| \leq |t| + 1$ ,  $|V(\mathcal{P})| \leq 2|t| + 2$ , and every marked vertex in  $\mathcal{P}$  is a source.*

*Proof.* Induction on complexity of the term  $t$ . The terms 0, 1, and  $x \in X$  do not involve any operations, the corresponding circuits  $C_0, C_1, C_x$  satisfy the conditions  $|M(\mathcal{P})| \leq 1$ ,  $|V(\mathcal{P})| \leq 2$ , and have the property that every marked vertex is a source. Now, assume that the statement holds for terms  $t_1$  and  $t_2$ . Let  $t = f(t_1, t_2)$ , where  $f$  is an operation from  $\mathcal{L}_0$  and  $\mathcal{P}_1 = \tau(t_1)$ ,  $\mathcal{P}_2 = \tau(t_2)$ . Let  $\mathcal{P} = f(\mathcal{P}_1, \mathcal{P}_2)$  constructed by the appropriate algorithm from Section 7. Since every vertex in  $\mathcal{P}_2$  is a source it immediately follows from Algorithms 7.1 and 7.7 that

$$|M(\mathcal{P})| \leq |M(\mathcal{P}_1)| + |M(\mathcal{P}_2)| \text{ and } |V(\mathcal{P})| \leq |V(\mathcal{P}_1)| + |V(\mathcal{P}_2)|$$

and every vertex in  $\mathcal{P}$  is a source. Therefore,  $|M(\mathcal{P})| \leq |t_1| + 1 + |t_2| + 1 = |t| + 1$  and  $|V(\mathcal{P})| \leq 2|t_1| + 2 + 2|t_2| + 2 = |t| + 2$ . Moreover, the circuit  $\mathcal{P}$  in both cases is computed in linear time in  $|t|$ .  $\square$

In contrast to Theorem 8.6 we construct in Section 9.2 a sequence of terms  $\{t_i\}$  with multiplication in the language such that the size of  $\tau(t_i)$  grows exponentially.

## 8.2 Power representation of integers

Let  $\mathcal{C}_{norm}$  be the set of all constant normal power circuits up to isomorphism (so  $\mathcal{C}_{norm}$  consists of the equivalence classes of isomorphic normal power circuits). Every operation  $f \in \mathcal{L}$  induces a similar operation  $f$  on  $\mathcal{C}_{norm}$  defined for  $\mathcal{P}_1, \mathcal{P}_2 \in \mathcal{C}_{norm}$  as  $(\mathcal{P}_1, \mathcal{P}_2) \rightarrow Normal(f(\mathcal{P}_1, \mathcal{P}_2))$ . Define a map  $\lambda : \mathbb{Z} \rightarrow \mathcal{C}_{norm}$  such that  $\lambda(n)$  is the unique (up to isomorphism) normal power circuit representing  $n \in \mathbb{Z}$ . The next proposition follows directly from the definition of  $\lambda$  and the results on normal power circuits.

**Proposition 8.4.** The following hold:

- the map  $\lambda$  defines an isomorphism of  $\mathcal{L}$ -structures

$$\lambda : \langle \mathbb{Z}; +, -, *, x \cdot 2^y, x \cdot 2^{-y}, 0, 1 \rangle \rightarrow \langle \mathcal{C}_{norm}; +, -, *, x \cdot 2^y, x \cdot 2^{-y}, 0, 1 \rangle.$$

- If  $t$  is a closed term in  $\mathcal{L}$  which gives a number  $n \in \mathbb{Z}$  then  $\lambda(n) = Norm(\tau(t))$ .

Let  $L_0 = \{+, -, x \cdot 2^y, x \cdot 2^{-y}\}$  and  $\mathcal{T}_{L_0}$  as above. The next algorithm solves the term realization problem for  $\mathcal{C}_{norm}$ .

**Algorithm 8.5.** (*Term realization for  $\mathcal{C}_{norm}$* )

INPUT. Let  $t(x_1, \dots, x_k) \in \mathcal{T}_L$  be a term in variables  $X_k = \{x_1, \dots, x_k\}$  and  $\eta : X_k \rightarrow \mathcal{C}_{norm}$  an assignment of variables.

OUTPUT. A circuit  $\mathcal{P}_t = t(\eta(X_k))$  if it is defined in  $\mathbb{Z}_{\mathcal{L}}$ . *Failure* otherwise.

COMPUTATIONS.

- (A) For every subterm  $u$  of  $t$  compute a reduced power circuit  $\mathcal{P}'_u$  realizing  $u$  as follows:

- If  $u$  is a term 0 then  $\mathcal{P}'_u = C_0$ .
- If  $u$  is a term 1 then  $\mathcal{P}'_u = C_1$ .
- If  $u$  is a term  $x \in X$  then  $\mathcal{P}'_u = \eta(x)$ .
- If  $u$  is a term  $u = f(u_1, u_2)$  where  $f$  is an operation from  $\mathcal{L}$  then apply Algorithm 7.1 or Algorithm 7.7 to circuits  $\mathcal{P}'_{u_1}$  and  $\mathcal{P}'_{u_2}$  (we assume they are already constructed). Reduce and denote the result by  $\mathcal{P}'_u$ .

If  $\mathcal{P}'_u$  does not represent an integer then output *Failure*.

- (B) Compute the normal power circuit  $\mathcal{P}_t$  equivalent to  $\mathcal{P}'_t$  (use Theorem 6.3).

- (E) Output the  $\mathcal{P}_t$ .

We summarize the discussion above in the following

**Theorem 8.6.** *Let  $t \in \mathcal{T}_{\mathcal{L}}$  and  $\eta : X \rightarrow \mathcal{C}_{norm}$  an assignment of variables. Algorithm 8.5 determines whether  $t(\eta(X))$  is defined in  $\mathbb{Z}_{\mathcal{L}}$  or not; and if defined then it produces the normal circuit  $\mathcal{P}_t$  which represent the number  $t(\eta(X))$ .*

The next result shows that the Algorithm 8.5 is of polynomial time on terms from  $\mathcal{T}_{L_0}$ . For a term  $t(X) \in \mathcal{T}_{L_0}$  and a variable  $x \in X$  define  $\sigma_x(t)$  to be the number of times the variables  $x$  occurs in  $t$ . Similarly, define  $\sigma_0(t)$  and  $\sigma_1(t)$  to be the number of occurrences of the constants 0 and 1 in  $t$ , respectively.

**Theorem 8.7** (Complexity of term realization). *Let  $t(X) \in \mathcal{T}_{L_0}$  and  $\eta : X \rightarrow \mathcal{C}_{norm}$  an assignment of variables. Let  $\mathcal{P}_t$  be the output of Algorithm 8.5. Then*

$$|M(\mathcal{P}_t)| \leq \sum_{x \in X} \sigma_x(t) \cdot |M(\eta(x))| + \sigma_1(t).$$

$$|V(\mathcal{P}_t)| \leq 2(|t| + 1) \left( \sum_{x \in X} \sigma_x(t) \cdot |V(\eta(x))| + 2\sigma_1(t) + \sigma_0(t) \right)$$

and Algorithm 8.5 terminates in

$$O \left( |t|^4 \cdot \left( \sum_{x \in X} \sigma_x(t) \cdot |V(\eta(x))| + 2\sigma_1(t) + \sigma_0(t) \right)^3 \right)$$

steps.

*Proof.* Following Algorithm 8.5 by induction on complexity of a subterm  $u$  we prove that

$$|M(\mathcal{P}'_u)| \leq \sum_{x \in X} \sigma_x(u) \cdot |M(\eta(x))| + \sigma_1(u). \quad (4)$$

$$|V(\mathcal{P}'_u)| \leq (|u| + 1) \left( \sum_{x \in X} \sigma_x(u) \cdot |V(\eta(x))| + 2\sigma_1(u) + \sigma_0(u) \right). \quad (5)$$

Indeed, the bounds (4) and (5) clearly hold for the elementary terms 0, 1, and  $x$ . If  $u = f(u_1, u_2)$  where  $f \in \mathcal{L}_0$  then one of the Algorithms 7.1 or 7.7 (depending on  $f$ ) produces a circuit  $\mathcal{P}$  such that  $\mathcal{E}(\mathcal{P}) = f(\mathcal{E}(\mathcal{P}'_{u_1}), \mathcal{E}(\mathcal{P}'_{u_2}))$ . For every such  $f$  we have

$$|M(\mathcal{P})| \leq |M(\mathcal{P}'_{u_1})| + |M(\mathcal{P}'_{u_2})| \quad (6)$$

$$|V(\mathcal{P})| \leq |V(\mathcal{P}'_{u_1})| + |V(\mathcal{P}'_{u_2})| + |M(\mathcal{P}'_{u_1})|. \quad (7)$$

Reducing the circuit  $\mathcal{P}$  to  $\mathcal{P}'_u$  does not increase the number of marked vertices, hence (6) holds for  $\mathcal{P}'_u$ . The inequality (6) immediately implies (4). The reduction process can introduce one auxiliary vertex, but since both  $\mathcal{P}_{u_1}$  and  $\mathcal{P}_{u_2}$  have a zero vertex, the bound (7) also holds for  $|V(\mathcal{P}'_u)|$ . Now, it

follows from (7) that every operation increases the number of marked vertices by at most  $|M(\mathcal{P}'_u)|$ , which is bounded in view of (4) by the number  $\sum_{x \in X} \sigma_x(u) \cdot |V(\eta(x))| + 2\sigma_1(u) + \sigma_0(u)$ . Thus the inequality (5) holds.

Finally, we use Theorem 6.3 to compute the normal circuit for  $\mathcal{P}'_t$ . That increases the total number of vertices by up to a factor of 2 and does not increase the number of marked vertices. Hence the required bounds for  $|V(\mathcal{P}_t)|$  and  $|M(\mathcal{P}_t)|$  follow.

The Algorithm 8.5 performs  $|t|$  reductions. By Proposition 5.17 the complexity of reducing a circuit  $\mathcal{P}$  requires  $O(|V(\mathcal{P})|^3)$  steps. Using the bound (5) we obtain the required bound on the time complexity of Algorithm 8.5, which finishes the proof.  $\square$

**Corollary 8.8.** Let  $t(X) \in \mathcal{T}_{L_0}$  and  $\eta : X \rightarrow \mathbb{Z}$  an assignment of variables. There exists an algorithm which determines if  $t(\eta(X))$  is defined in  $\mathbb{Z}_{\mathcal{L}}$  (or  $\tilde{\mathbb{Z}}$ ) or not; and if defined it then produces the normal circuit  $\mathcal{P}_t = \lambda(t(\eta(X))) \in \mathcal{C}_{norm}$ . The algorithm has time complexity

$$O\left(|t|^4 \cdot \left(\sum_{x \in X} \sigma_x(t)(s_x + 2) + 2\sigma_1(t) + \sigma_0(t)\right)^3\right)$$

where  $s_x = \lceil \log_2(|\eta(x)| + 1) \rceil$ .

*Proof.* The required algorithm first constructs the normal circuits  $\mathcal{P}_x$  representing  $\eta(x)$  for every  $x \in X$  and then applies Algorithm 8.5. By Theorem 6.2 the time complexity of computing  $\mathcal{P}_x$  is  $O(\log_2(s_x) \log_2 \log_2(s_x))$  and  $|V(\mathcal{P}_x)| \leq s_x + 2$ . Application of Theorem 8.7 finishes the proof.  $\square$

### 8.3 Quantifier-free formulas in exponential algebra

In this section we study the quantifier-free theory of the  $\mathcal{L}_0$ -structure

$$\tilde{\mathbb{Z}} = \langle \mathbb{Z}; +, -, x \cdot 2^y, x \cdot 2^{-y}, \leq, 0, 1 \rangle$$

with all the constant from  $\mathbb{Z}$  in the language. To this end we extend the language  $\mathcal{L}_0$  to  $\mathcal{L}_0^{const}$  by adding a constant symbol  $n$  for every  $n \in \mathbb{Z}$ . The structure  $\tilde{\mathbb{Z}}$  naturally extends to a structure  $\tilde{\mathbb{Z}}_{const}$  in the language  $\mathcal{L}_0^{const}$ . Since  $\mathcal{L}_0^{const}$  is an infinite language the complexity of algorithmic problems in  $\tilde{\mathbb{Z}}_{const}$  depends on how we present the data, in this case, the constants  $n \in \mathbb{Z}$ . We assume here that all the integers  $n \in \mathbb{Z}$  are given in their binary forms. Of course, since every integer  $n$  can be presented as a closed term in the structure  $\langle \mathbb{Z}, +, -, 0, 1 \rangle$ , every term  $t$  in the structure  $\tilde{\mathbb{Z}}_{const}$  can be presented by a term  $t'$  in the structure  $\tilde{\mathbb{Z}}$ , but in this case the length of the term  $t'$  can grow exponentially in the length of  $t$ . In such event one would allow too much of leeway to himself (when working on complexity problems) by representing integers in the unary form, and the results would be weaker.

**Theorem 8.9.** *The quantifier-free theory of the structure  $\tilde{Z}_{const}$  is decidable in polynomial time.*

*Proof.* A quantifier free formula in  $\tilde{Z}_{const}$  is a formula of the type  $t_1(X) \diamond t_2(X)$  where  $t_1, t_2 \in \mathcal{T}_{L_0}$  and  $\diamond \in \{\leq, =\}$ . To determine if it holds in  $\tilde{Z}_{const}$  it suffices to compute the normal power circuit representing the term  $t_1 - t_2$  and use Proposition 7.11 to compare the value  $\mathcal{E}(\mathcal{P})$  with 0. Both operations have polynomial time complexity in terms of the size of the formula, hence the result.  $\square$

**Corollary 8.10.** The quantifier-free theory of the structure  $\tilde{Z}$  is decidable in polynomial time.

**Corollary 8.11.** The quantifier-free theory of the structure  $\tilde{N} = \langle \mathbb{N}; +, -, x \cdot 2^y, x \cdot 2^{-y}, \leq, 0, 1 \rangle$  is decidable in polynomial time.

## 9 Some inherent difficulties in computing with power circuits

In this section we demonstrate that a product of power circuits may result in a power circuit whose size may grow exponentially in the size of the factors. We also show that solving some linear equations in power circuits may take super-exponential time.

### 9.1 Division by 3

For each natural  $i$  consider a number

$$N_i = 2^{2^i} + 2^{2^{(i-1)}} + \dots + 2^2 + 2^0 = \frac{4^{i+1} - 1}{3}.$$

The binary sum above is compact, so by Lemma 2.8 it is a shortest binary sum decomposition of  $N_i$ . Hence any other binary decomposition of  $N_i$  contains at least  $i+1$  terms. This implies that any power circuit  $\mathcal{P}_i$  representing the number  $N_i$  contains at least  $i+1$  vertices. Now, pick

$$i = \text{tower}_2(j) = 2^{2^{\dots^2}} \text{ } j \text{ times}.$$

Then  $3N_i = 4^{i+1} - 1$  and there exists a circuit, say  $\mathcal{P}_j$ , on  $j+1$  vertices representing  $4^{i+1} - 1$ . This follows that the linear equation  $3x = \mathcal{P}_j$  has a solution  $\mathcal{P}_i$  in the power circuit arithmetic, but any power circuit that gives a solution of this equation has at least  $i = \text{tower}_2(j)$  vertices. This proves the following proposition.

**Proposition 9.1.** The worst case complexity of solving a linear equation  $3x = \mathcal{P}$  in power circuits ( $\mathcal{P}$  is a constant and  $x$  is a variable over the set of power circuits) is super-exponential.

We conclude this section with an observation that prime factorization of numbers given by power circuits can be super-exponential.

## 9.2 Power circuits and multiplication

In this section we demonstrate some inherent difficulties when dealing with products of power circuits (the size of the resulting circuit grows exponentially).

For  $n \in \mathbb{N}$  define a power circuit  $\mathcal{P}_n = (\mathcal{P}_n, \mu, M, \nu)$ , where

- $\mathcal{P}_n = (V_n, E_n)$  and  $V_n = \{0, \dots, n\}$  and  $E_n = \{(i, i-1) \mid i = 1, \dots, n\} \subset V_n \times V_n$ ;
- $\mu \equiv 1$ ;
- $M = \{1, n\}$ ;
- $\nu(1) = \nu(n) = 1$ .

Clearly,  $\mathcal{E}(\mathcal{P}_n) = \text{tower}_2(n-1) + 1$  and  $|\mathcal{P}_n| = n+1$ . The product  $\mathcal{P}_4 \cdot \dots \cdot \mathcal{P}_n$  represents the number

$$\begin{aligned} \prod_{i=4}^n (\text{tower}_2(i-1) + 1) &= \sum_{\sigma \in \{0,1\}^{n-3}} \left( \prod_{4 \leq i \leq n, \sigma_{i-3}=1} \text{tower}_2(i-1) \right) \\ &= \sum_{\sigma \in \{0,1\}^{n-3}} \left( \prod_{4 \leq i \leq n, \sigma_{i-3}=1} 2^{\text{tower}_2(i-2)} \right) = \sum_{\sigma \in \{0,1\}^{n-3}} 2^{s_\sigma} \end{aligned}$$

where

$$s_\sigma = \sum_{4 \leq i \leq n, \sigma_{i-3}=1} \text{tower}_2(i-2).$$

The binary sum  $\sum_{\sigma \in \{0,1\}^{n-3}} 2^{s_\sigma}$  is compact and hence by Lemma 2.8 involves the least number  $2^{n-3}$  of terms. Therefore the product  $\mathcal{P}_4 \cdot \dots \cdot \mathcal{P}_n$  can not be represented by a power circuit of size less than  $2^{n-3}$ .

## 10 Open Problems

In this section we state some interesting algorithmic problems for exponential algebras.

**Problem 10.1.** Can one develop a robust theory of power circuits when  $\mathbb{Z}$  is replaced by  $\mathbb{Q}$ ? or  $\mathbb{R}$ ?

Here the main concern is the reduction algorithm.

**Problem 10.2.** 1) Is the quantifier-free theory of the standard high-school arithmetic  $\mathbb{N}_{HS}$  polynomial time decidable (with all constants from  $\mathbb{N}$  in the language)?

2) Is the equational theory of  $\mathbb{N}_{HS}$  polynomial time decidable?

The example in Section 9.2 demonstrates that power circuits in the structure  $\mathbb{N}_{HS}$  do not allow fast manipulations that involve arbitrary multiplications. Nevertheless, it might be that there are some other means to approach the problem.

**Problem 10.3.** Is the existential theory of  $\tilde{N} = \langle \mathbb{N}_{>0}; +, x \cdot 2^y, \leq, 1 \rangle$  decidable? Is the Diophantine problem decidable?

**Problem 10.4.** What is the time complexity of the the problem of finding a minimal (in size) constant power circuit representing a given natural number?

**Problem 10.5.** Is  $\tilde{N}$  automatic?

## References

- [1] A. V. Aho and J. D. Ullman, *Transformations on straight line programs*. Proceedings of the second annual ACM symposium on Theory of computing, Annual ACM Symposium on Theory of Computing STOC '70, pp. 136–148. ACM, New York, 1970.
- [2] S. Arora and B. Barak, *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.
- [3] G. Baumslag, *A non-cyclic one-relator group all of whose finite factor groups are cyclic*, J. Australian Math. Soc. 10 (1969), pp. 497–498.
- [4] D. Bini and V. Y. Pan, *Polynomial and matrix computations (vol. 1): fundamental algorithms*. Birkhauser Verlag, Basel, Switzerland, Switzerland, 1994.
- [5] A. Blumensath and E. Gradel, *Automatic structures*. 15th symposium on logic in computer science, Logic in Computer Science, 2000, pp. 51–62. LICS 2000, 2000.
- [6] R. Book and F. Otto, *String-rewriting systems*, Texts and monographs in computer science. Springer, 1993.
- [7] P. Burgisser, M. Claussen, and M. Amin Shokrollahi, *Algebraic complexity theory*. Springer, Berlin, 1997.
- [8] R. Dedekind, *What are numbers and what should they be?*. Research Institute for Mathematics, 1995.
- [9] S. M. Gersten, *Dehn functions and l1-norms of finite presentations*. Algorithms and Classification in Combinatorial Group Theory, pp. 195–225. Springer, Berlin, 1992.
- [10] Y. Gurevich, *Equational theory of positive numbers with exponentiation is not finitely axiomatizable*, Ann. Pure Appl. Logic 49 (1990), pp. 1–30.



- [11] B. R. Hodgson, *Theories decidables par automate fini*. Phd thesis, University of Montreal, 1976.
- [12] I. Kapovich and P. Schupp, *Genericity, the Arzhantseva-Ol'shanskii method and the isomorphism problem for one-relator groups*, Math. Ann. 331 (2005), pp. 1–19.
- [13] B. Khossainov and A. Nerode, *Automatic presentations of structures*, Lecture Notes in Computer Science 960 (1995), pp. 367–392.
- [14] M. Lohrey, *Word problems on compressed words*. Automata, languages and programming, Lecture Notes in Computer Science 3142, pp. 906–918. Springer-Verlag, Berlin, 2004.
- [15] A. Macintyre, *The laws of exponentiation*. Model theory and arithmetic, Lecture Notes in Mathematics 890, pp. 185–197. Springer, 1981.
- [16] ———, *Exponential Algebra*. Logic and Algebra. Proceedings of the international conference dedicated to the memory of Roberto Magari, Lect. Notes Pure Appl. Math. 180, pp. 191–210. Springer, 1991.
- [17] ———, *Schanuel's Conjecture and free exponential rings*, Ann. Pure Appl. Logic 51 (1991), pp. 241–246.
- [18] A. Macintyre and A. Wilkie, *On the decidability of the real exponential field*. Kreiseliana: About and Around Georg Kreisel, pp. 441–467. AK Peters, Ltd, 1996.
- [19] A. I. Malcev, *Constructible Algebras*, Uspekhi Mat. Nauk 16 (1961), pp. 3–60.
- [20] A. G. Miasnikov, A. Ushakov, and Dong Wook Won, *Word problem in Baumslag-Gersten group is polynomial time decidable*, to appear.
- [21] W. Plandowski, *Testing equivalence of morphisms on context-free languages*. AlgorithmsESA 94 (Utrecht), Lecture Notes in Computer Science 855, pp. 460–470. Springer-Verlag, Berlin, 1994.
- [22] A. N. Platonov, *Isoparametric function of the Baumslag-Gersten group*, (Russian) Vestnik Moskov. Univ. Ser. I Mat. Mekh. (2004), pp. 12–17.
- [23] M. Rabin, *Computable algebra, general theory and theory of computable fields*, T. Am. Math. Soc. 94 (1960), pp. 341–360.
- [24] D. Richardson, *A solution of the identity problem for integral exponential functions*, Z. Math. Logik Grundlag. Math. 15 (1969), pp. 333–340.
- [25] ———, *Roots of real exponential functions*, Bull. London Math. Soc. (2) 28 (1983), pp. 46–56.

- [26] ———, *The elementary constant problem*. International Conference on Symbolic and Algebraic Computation 1992, ISSAC, pp. 108–116. ACM, New York, NY, USA, 1992.
- [27] S. Rubin, *Automata presenting structures: A survey of the finite string case*, Bull. Symbolic Logic 14 (2008), pp. 169–209.
- [28] S. Schleimer, *Polynomial-time word problems*, Comment. Math. Helv. 83 (2008), pp. 741–765.
- [29] C. E. Shannon, *The synthesis of two-terminal switching circuits*, BELL Syst. Tech. J. 28 (1965), pp. 59–98.
- [30] V. Strassen, *Gaussian Elimination is not Optimal*, Numer. Math. 13 (1969), pp. 354–356.
- [31] ———, *Algebraic complexity theory*. Handbook of Theoretical Computer Science, Lecture Notes in Computer Science, Volume A, J.van Leeuwen ed., pp. 633–673. Elsevier, 1990.
- [32] L. van den Dries, *Exponential rings, exponential polynomials and exponential functions*, Pac. J. Math. 113 (1984), pp. 51–66.
- [33] C. M. Weinbaum, *On relators and diagrams for groups with one defining relator*, Illinois J.Math. 16 (1972), pp. 308–322.
- [34] A. J. Wilkie, *On exponentiation a solution to Tarski’s high school algebra problem*, Quad. Mat. 6 (2000), pp. 107–129.
- [35] Z. Zilber, *Pseudo-exponentiation on algebraically closed fields of characteristic zero*, Ann. Pure Appl. Logic 132 (2004), pp. 67–95.